VSB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Measurement and Control

Diploma Thesis

Digital Circuit Emulator with programmable Logic

Ostrava 2010 Ibrahim Salem Jahan

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Measurement and Control

Diploma Thesis Assignment

Student: **Ibrahim Salem Jahan**

Study Programme: N2649 Electrical Engineering

Study Branch: 2601T004 Measurement and Control Engineering

Title: Digital circuits emulator with programmable logic

Digital circuits emulator with programmable logic

Description:

1. Introduction to programmable logic and VHDL language.
2. Analysis of logic functions emulation possibilities on the FPGA platform.
3. Design of selected logic functions in VHDL.
4. Implementing the logic design into Spartan3 Starter Kit board.
5. Verification and experimental testing of the HW Emulator.
6. Results evaluation and conclusion.

References:

- PARNELL, K., MEHTA, N. Programmable Logic Design Quick Start Handbook. Xilinx Inc., 2003.
- ASHENDEN, P. The Designer's Guide to VHDL. Morgan Kaufmann Publishers, 1998. ISBN 1-55860-270-4.
- Xilinx, Inc.: Spartan-3 FPGA Starter Kit Board User Guide. Available on: http://www.xilinx.com/support/documentation/boards_and_kits/ug130.pdf [cit. 2009-11-13].

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Ing. Vladimír Kašík, Ph.D.**

Date of issue: 20.11.2009

Date of submission: 07.05.2010

doc. Ing. Jiří Koziorek, Ph.D.
Head of Department

prof. Ing. Ivo Vondrák, CSc.
Dean of Faculty

Declaration of Authorship

I hereby confirm that I have authored this thesis independently and without use of other than the indicated resources.
All passages, which are literally or in general manner taken out of publications or other sources market as such.

.....

Ibrahim Salem Jahan

Date of submission of thesis: 7/ 5/ 2010

Acknowledgements

I wish thank my guide Ing. Vladimir Kasik, Ph.D. , for his valuable guidance. I also wish to thank Doc. Ing. Jiri Koziorek, Ph.D. , and Prof. Ing. Vilem Srovnal, CSc. for their help in chose exactly my diploma work according to my field, and according to also modern specializations.

List of used the symbols and abbreviations

ASIC	Application Specific Integrated Circuit
BCD	Binary-Coded Decimal
CE	Clock Enable
CLB	Configurable Logic Block
CLR	Clear
CLK or CK	Clock Signal
CPLD	Complex Programmable Logic Devices
CRT	Cathode Ray Tube
DCM	Digital Clock Manager
DIN	Data Input
DDR	Double Data Rate Register
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FF or FFs	Flip-Flops
HDL	Hardware Description Language
HEX	Hexadecimal
I O	Input, Output
I /O	Input / Output
IOB	Input Output Block
ISP	In System Programming
JTAG	Joint Test Advisory Group
LED	Light Emitter Diode
LSB	Least Significant Bit
LUT	Lock Up Table
MSB	Most Significant Bit
MUX	Multiplexer
NAND	Not And
PACE	Pinout and Area Constrains Editor
PAL	Programmable Array Logic
PRE	Preset
PCB	Printed Circuit Board
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
SRAM	Static Random Access Memory
SW	Switch
TTL	Transistor Transistor Logic
UCF	User Constraints File
VCCO	Voltage Current Controlled Oscillator
VGA	Video Graphics Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language
XOR	Exclusive OR

Abstract

For big development that occur in the world now to take to abbreviate the micro computers and logical devices such as processors and memories etc , addition to used one chip for execution several processes, hence invent the FPGA. This research explain, how use the FPGA chip in building and construction several logic circuits inside one FPGA chip, by means of writing the logic circuits by VHDL Language and download this circuits that written by VHDL Language inside FPGA chip, rather than used several Integral circuits in last . The FPGA chip abbreviate the time, decrease number of ICs that used and decrease also the external connecters that used in last (In ASIC chip). The FPGA can use such as processor, or memory, or microcontroller at same time.

Contents

1 INTRODUCTION 1

1.1 THE HISTORY OF PROGRAMMABLE LOGIC 1
1.2 COMPLEX PROGRAMMABLE LOGIC DEVICES (CPLD) 1
1.3 FILED PROGRAMMABLE GATE ARRAY (FPGA) 2
1.3.1 History of FPGA 2
1.3.2 What the FPGA? 3
1.3.3 Architecture of FPGA 3
1.3.4 How FPGAs Work 6
1.3.5 Features of FPGA 7
1.3.6 Who Make The FPGA? 8
1.3.7 FPGA Power 8
1.3.8 FPGAs Pins 8
1.3.9 Applications of FPGA 9
1.3.10 FPGA Design and Programming 9
1.3.11 Steps Design of FPGA Chip 9
1.3.12 FPGA Comparison 10
1.4 WHAT IS DIFFERENCE BETWEEN FPGA AND CPLD? 10
1.5 VHDL LANGUAGE 11
1.5.1 Definition 11
1.5.2 About VHDL 11
1.5.3 Basic Structure of a VHDL file 11
1.5.4 Some Simple Code Examples by VHDL Language 12

2. DESCRIPTION OF BASIC LOGIC FUNCTIONS 14

2.1 INTRODUCTION 14
2.2 THE MAIN LOGIC GATES AND OTHER GATES 14
2.3 FULL ADDER (FA) 14
2.4 DIGITAL COMPARISON 15
2.5 SHIFT REGISTER 16
2.6 MULTIPLEXERS 16
2.7 FULL SUBTRACTER 16
2.8 THE FLIP-FLOPS 17
2.8.1 JK Flip-Flop 17
2.8.2 D Flip-Flop 17
2.9 DECODER 18
2.10 SYNCHRONOUS COUNTER 18
2.11 TRAFFIC LIGHTS CONTROL 19
2.12 BINARY MULTIPLIER 19
2.13 MEMORY (RAM) 19
2.13.1 Block Ram 19
2.13.2 Distributed RAM 20
2.14 DEBOUNCE CIRCUIT 20
2.15 PRIORITY ENCODER CIRCUIT 20
2.16 DIGITAL STOPWATCH 21
2.17 BARREL SHIFTER 21
2.18 LD LATCH 22
2.19 DOUBLE DATA REGISTER (DDR REGISTER) 22
2.20 DIGITAL ACCUMULATOR 23

3. DESIGN OF LOGIC EMULATOR STRUCTURE 24

3.1 Introduction 24
3.2 The Tools that used for Execute Logic Circuits 24
3.3 Spartan 3 FPGA Board 24
3.4 Diagram of Digital Circuit Emulator with Programmable Logic 25
3.5 Design of Some Logic Functions by VHDL Language 26
3.5.1 Simple Logic Gates 26
3.5.2 Multiplexers Circuit 27
3.5.3 Decoder Circuit 27
3.5.4 Binary Multiplier Circuit 29
3.5.5 D Flip-Flop Circuit 30
3.5.6 Synchronous Counter Circuit 31
3.5.7 Debounce Circuit 32
3.5.8 Shift Register Circuit 32
3.5.9 Digital Comparator Circuit 33
3.5.10 Digital Accumulator 34
3.5.11 Latch Circuit 35
3.5.12 Priority Encoder Circuit 35
3.5.13 Barrel Shifter Circuit 36
3.5.14 Stop Watch Timer Circuit 37
3.5.15 Traffic Lights Control Circuit 38
3.5.16 Double Data Rate Register (DDR Register) 43
3.5.17 Distributed RAM Memory 44

4. IMPLEMENTING THE LOGIC DESIGN INTO SPARTAN-3 STARTER KIT BOARD 46

4.1 Introduction 46
4.2 User Constraints 46
4.3 Synthesis 48
4.4 Implement the Design 49
4.4.1 Translate 49
4.4.2 Map 49
4.4.3 Place & Route 50
4.5 Generate Program File 51
4.6 The Area Inside FPGA Chip That Used for Implementation The Design 52

5. VERIFICATION AND TESTING OF THE HARDWARE EMULATOR 53

5.1 Introduction 53
5.1 Testing and Verification of Function No. 11 The Binary Counter Circuit 53
5.2 Testing and Verification of Function No. 6 The Barrel Shifter Circuit 54
5.3 Testing and Verification of Function No. 9 The D Latch Circuit 55
5.4 Testing and Verification of Function No. 3 The Binary Multiplier Circuit 56
5.5 Testing and Verification of Function No. 12 - The Distributed RAM with VGA Display 56
5.6 Testing and Verification of Function No. 15 - The Traffic Lights Control with VGA Display 57

6. CONCLUSION 58

6.1 FPGA Technology Versus Conventional Logic ICs 58
6.2 Conclusion 59

7. REFERENCES 60

APPENDIX 1 62

1 Introduction

1.1 The History of Programmable Logic

By the late 1970s, standard logic devices were all the rage, and printed circuit boards were loaded with them. Then someone asked what if we gave designer the ability to implement different interconnections in a bigger device? This would allow designers to integrate many standard logic devices into one part.

To offer the ultimate in design flexibility, Ron Cline from Signetics (which was later purchased by Phlips and then eventually Xilinx) came up with the idea of two programmable planes. These two planes provided any combination of AND and OR gates, as well as sharing of AND terms across multiple ORs. [1]

1.2 Complex Programmable Logic Devices (CPLD)

A complex programmable logic device (CPLD) is a programmable logic device with complexity between that of PALs and FPGAs, and architectural features of both. The building block of a CPLD is the macro cell, which contains logic implementing disjunctive normal form expressions and more specialized logic operations. [2]

Why Use a CPLD?

CPLD enable ease of design, lower development costs, more product revenue for money, and the opportunity to speed your product to market. [1]

Ease of Design: CPLDs offer the simplest way to implement a design. Once a design has been described, by schematic and / or HDL entry, you simply use CPLD development tools to optimize, fit, and simulate the design.

Lower Development Costs: CPLDs offer very low development costs. Because CPLDs are re-programmable, you can easily and very inexpensively change your design. This allows you to optimize your design and continue to add new features to enhance your products.

More Product Revenue: CPLDs offer very short development cycles, which means your products get to market quicker and begin generating revenue sooner. Because CPLDs are re-programmable, products can be easily modified using ISP over the Internet. This in turn allows you to easily introduce additional features and quickly generate new revenue.

Reduced Board Area: CPLDs offer a high level of integration (that is, a large number of system gates per area) and are available very small form factor packages. This provides the perfect

solution for designer whose products which must fit into small enclosures or who have a limited amount of circuit board space to implement the logic design.

Xilinx CoolRunner CPLDs are available in the last chip scale packages. For example, the CP56 CPLD has a pin pitch of 0.5 mm and is a mere 6 mm in 6mm in size, marking it ideal for small, low power product.

Cost of Ownership: Cost of Ownership can be defined as the amount it costs to maintain, fix, or warranty a product. For instance, if a design change requiring hardware rework must be made to a few prototypes, the cost might be relatively small. [1]

1.3 Filed Programmable Gate Array (FPGA)

In 1985, a company called Xilinx introduced a completely idea: combine the user control and time to market of PLDs with the densities and cost benefits of gate arrays.

Customer liked it and the FPGA was born. Today Xilinx is still the number one FPGA vendor in the world.

An FPGA is a regular structure of logic cells (or modules) and interconnect, which is under your complete control. This means that you can design, program, and make changes to your circuit whenever you wish.

With FPGA now exceeding the 10 million gate limit (the Xilix Virtex –II FPGA is the current record holder). [1]

1.3.1 History of FPGA

In the late 1980s the Naval Surface Warfare Department funded an experiment proposed by Steve Casselman to develop a computer that would implement 600,000 reprogrammable gate. Casslman was successful and the system was awarded a patent in 1992. Some of the industry's foundational concepts for programmable logic arrays, gates, and logic blocks are founded in patents awarded to David W, in 1985. Xilinx Co Founders, invented first commercially viable FPGA in 1985, the XC2064. The xc2064 had programmable gates and programmable interconnects between gates. The XC2064 has 64 configurable logic block (CLBs), with two 3-input lookup tables (LUTs). Xilinx continued growing from 1985 to the mid 1990s, when the competitors sprouted up. By 1993 Actel Co was serving about 18% of market. In the early 1990s, FPGAs, were primarily used in telecommunications and networking. By end of the decade, FPGAs found to consumer (automotive and industrial applications). In 1997, FPGAs got a glimpse of fame, when Adrian Thompson merged genetic algorithm technology and FPGAs to create a sound recognition device.[3]

1.3.2 What the FPGA?

The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnect from a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit. [4]

1.3.3 Architecture of FPGA

The most common FPGA architecture consists of an array of configurable logic blocks (CLBs), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array as figure 1.1.

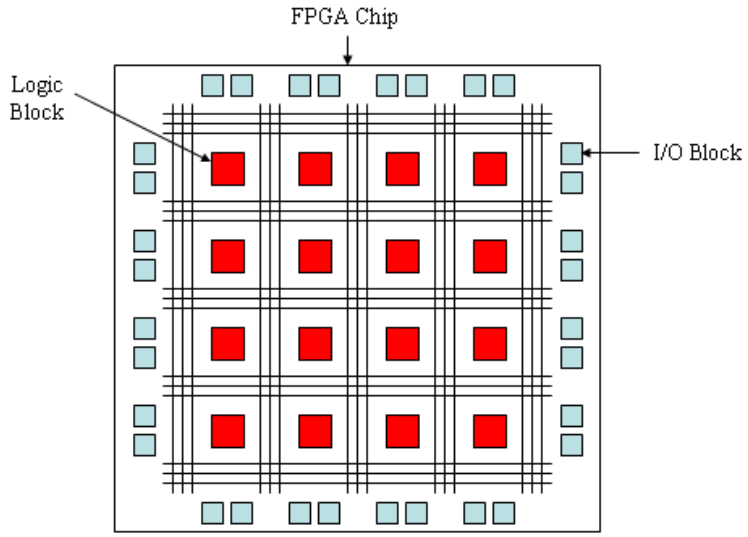


Figure 1.1 Structure of FPGA Chip

An application circuit must be mapped into an FPGA with adequate resources. While the number of CLBs and I/Os required is easily determined from the design, the number of routing tracks needed may vary considerably even among designs with the same amount of logic. (For example, a crossbar switch requires much more routing than a systolic array with the same gate count.) Since unused routing tracks increase the cost (and decrease the performance) of the part without providing any benefit, FPGA manufacturers try to provide just enough tracks so that most

designs that will fit in terms of LUTs and IOs can be routed. This is determined by estimates such as those derived from Rent's rule or by experiments with existing designs.

A classic FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop , as figure 1.2. In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance.

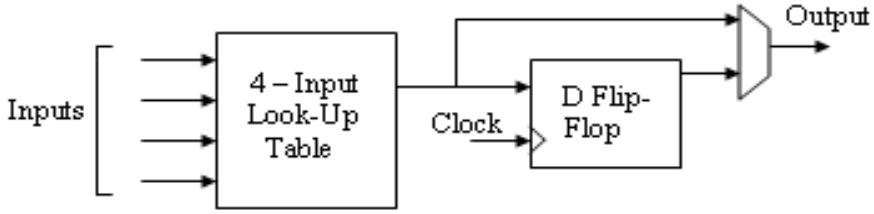


Figure 1.2 FPGA Logic block

There is only one output, which can be either the registered or the unregistered LUT output. The logic block has four inputs for the LUT and a clock input. Since clock signals (and often- other high-fanout signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed.

For this example architecture, the locations of the FPGA logic block pins are shown in Figure 1.3.

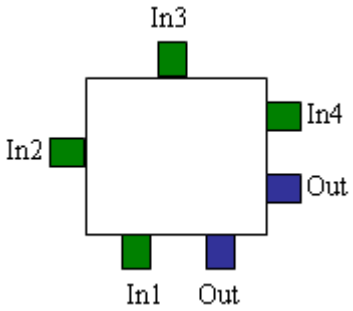


Figure 1.3 Logic Block Pin Locations

Each input is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block.

Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it.

Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it.

Generally, the FPGA routing is unsegmented. That is, each wiring segment spans only one logic block before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths can be constructed. For higher speed interconnect, some FPGA architectures use longer routing lines that span multiple logic blocks.

Whenever a vertical and a horizontal channel intersect, there is a switch box. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. In this switch box , a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. The figure 1.4 illustrates the connections in a switch box.

Modern FPGA families expand upon the above capabilities to include higher level functionality fixed into the silicon. Having these common functions embedded into the silicon reduces the area required and gives those functions increased speed compared to building them from primitives. Examples of these include multipliers, generic DSP blocks, embedded processors, high speed IO logic and embedded memories.

FPGAs are also widely used for systems validation including pre-silicon validation, post-silicon validation, and firmware development. This allows chip companies to validate their design before the chip is produced in the factory, reducing the time to market. [3]

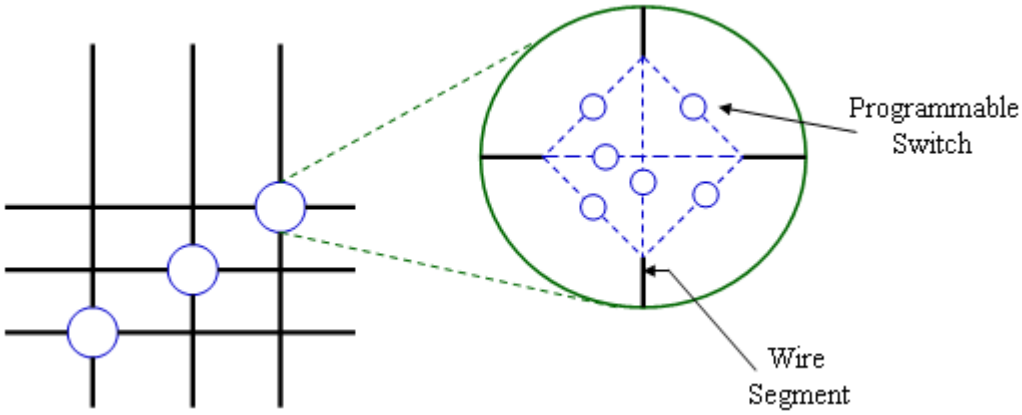


Figure 1.4 Switch box

Architecture of Spartan-3 family

The Architecture of Spartan-3 consists of five fundamental Programmable functional elements:

1. Configurable logic blocks (CLBs) contain RAM-based Look Up Table (LUTs) to implement logic and storage element that can be used as flip-flops or latches.
2. Input / Output Blocks (IOBs) control the flow of data between I/O pins and internal logic of the device. Each I/O Block supported bidirectional data flow plus 3- state operation.
3. Block RAM provides data store in the from of 18-Kbit dual port blocks.
4. Multiplier blocks accept two 18-bit binary numbers as inputs and calculate product.
5. Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals. As shown in figure 1.5 [5]

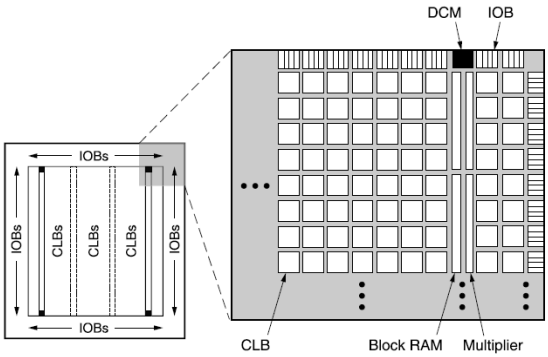


Figure 1.5 Architecture of Spartan-3 family [5]

1.3.4 How FPGAs Work

Logic Cells

FPGAs are built from one basic 'logic cell' , duplicated hundreds or thousands of them. A logic lookup table (LUT) , DFF and a 2 to 1 mux (to bypass the flipflop if desired). The LUT is lik a small RAM that can implement any logic function. It has typically a few inputs as figure 1.6, for example an AND gate with 3 inputs , whose results is then OR-ed with another input would fit in one 4-inputs LUT.[6]

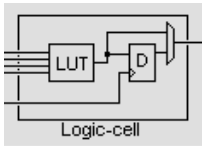


Figure 1.6 Logic Cell [6]

Interconnect

Each logic cell can be connected to other logic cell through interconnect resources (wires / muxes placed around the logic cell) . Each cell can do little, but with lost of them connected together , complex logic functions can be created as figure 1.7. [6]

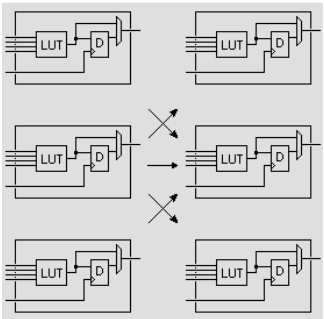


Figure 1.7 FPGA Interconnect [6]

IO- Cells

The interconnect wires also go to the boundary of the device where I/O cells are implemented and connected to the pins of the FPGAs, as figure 1.8. [6]

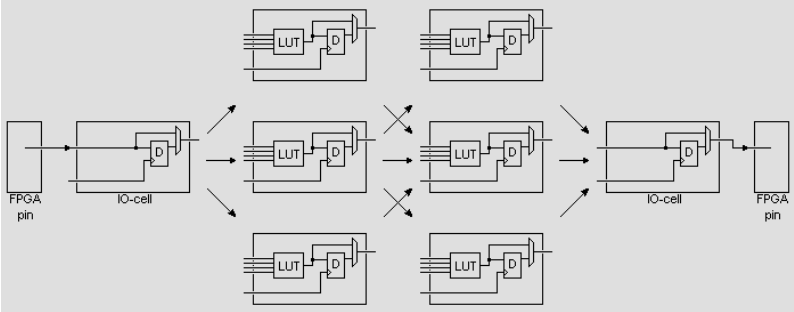


Figure 1.8 I/O Cells [6]

Dedicated Routing / Carry Chains

In general purpose interconnect resources , FPGAs have fast dedicated lines in between neighboring logic cells. The most common type of fast dedicated lines are ' carry chains' . Carry chains allow creating arithmetic functions (like counters and adders) efficiently (low logic usage & high operating speed), as figure 1.9. [6]

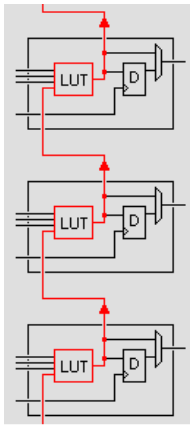


Figure 1.9 Dedicated Routing / Carry Chains [6]

1.3.5 Features of FPGA

1. Easy to design and detecting the defects and modified this defect in chip FPGA than chip ASIC.
2. Not needed to complex manufacture for execute electronic circuits with FPGA.
3. Design CPU for special purpose, because FPGAs easier to design, so the several companies that make the FPGAs , they are give free CPU on web page .

4. The chips FPGAs, they are change their function during period their work, so can put several designs inside FPGA chip, all this designs don't working at same time in FPGA chip, but some them work only when needed them.
5. The hardware designs of FPGA usually, give them by free. [7]

1.3.6 Who Make The FPGA?

There are several companies making FPGAs in the world. The first two (**Xilinx** and **Altera**) hold the bulk of the market. [6]
Xilinx and **Altera** are the current FPGA market leaders and long-time industry rivals. Together, they control over 80 percent of the market , with Xilinx alone representing over 50 percent.[3]
Xilinx also provides free Windows and Linux design software , while **Altera** provides free Windows tools; the Solaris and Linux tools are only available via a rental scheme.[3]
Other competitors include **Lattice** Semiconductor (flash, SRAM), **Actel** (antifuse, flash-based, mixed-signal), **SiliconBlue** Technologies (low power), **Achronix** (RAM based, 1.5GHz fabric speed), and **QuickLogic** (handheld focused CSSP, no general purpose FPGAs). [3]

1.3.7 FPGA Power

FPGAs usually required two voltages to operate a 'core voltage' an 'IO voltage'. Each voltage is provided through separate pins.

- The internal core voltage (called VCCINT here) is fixed (set by the model of FPGA that used). It is used to power the logic gates and flip-flops inside the FPGA. The value of voltage (3.3V, 2.5V, 1.8V, 1.5V, 1.2V).
- The IO voltage (called VCCIO here) is used to power the I/O blocks (pins) of the FPGA. That voltage should match what the other devices connected to the FPGA expect. [6]

The internal voltage is named 'VCC' for Xilinx and 'VCCINT' for Altera.
The IO voltage is named 'VCCO' for Xilinx and 'VCCIO' for Altera. [6]

1.3.8 FPGAs Pins

FPGAs chip have dedicated and user pins.
About 20% to 30% of the pins of an FPGA are 'dedicated pins', which means that they are hard coded to a specific function.
The dedicated pins fall into three categories:
Power pins: ground and core /IO power pins.
Configuration pins: used to download the FPGA.
Dedicated inputs, or clock pins: these are able to driver large nets inside the FPGA, suitable for clock or signal with large fan-outs.
The rest are user pins (called 'IOs', or 'I/Os', or 'user I/Os', or 'user IOs', or 'IO pins')
The IOs, they can be programmed to be inputs, outputs, or bidirectional.

Each "IO pins" is connected to an "IO cell" inside FPGA. The "IO cells" are powered by the VCCIO pins (IO power pins).
In new generation of FPGAs have a concept of "use IO banks". The IOs are split into groups, each having its own VCCIO voltage. That allows using the FPGA as a voltage translator device. [6]

1.3.9 Applications of FPGA

Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy and a growing rang of other area. [3]

1.3.10 FPGA Design and Programming

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The HDL might easier to work with large structures because it's possible to just specify them numerically rather than draw every piece by hand. But schematic entry can allow for easier visualization of a design.
Then, using an electronic design automation tool, a technology mapped netlist is generated. The netlist can then be fitted to the actual FPGA architecture using a process called Place and Rout . The user will validate the map, place and rout results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary file generated is used to (re) configure the FPGA.

The HDLs are VHDL and Veriolog, an attempt to reduce the complexity of design in HDLs, compared to the equivalent of assembly languages.

In typically design flow, an FPGA application developer will simulate the design at multiple stage throughout the design process. Initially the RTL description in VHDL or Verilog is simulated by creating test benches to simulate the system and abserve the results. Then, after the synthesis engine has mapped the design to a netlist, the netlist is translate to a gate level description where simulation is repeated to confirm the system proceeded without errors. Finally the design is laid out in the FPGA at which point propagation delays can be added and the simulation run again with these values back annotated onto the netlist. [3]

1.3.11 Steps Design of FPGA Chip

1. Definition the purpose of chip, definition the problem that need solution, and purpose of the chip, this step important before start the design.
2. Write the design, in this step will start write the design by draw schematic entry for circuit, but this method don't good for big designs. Or by write the design using VHD Language like VHDL or verilog.
3. Simulation ,the simulation used to detect the errors in design and repair them before programming.

4. Synthesis ,extraction of the components of circuit that described by VHD Language, for change this describe to electronic circuit , this method don't used when using the schematic entry.
5. Place and route , comparison the design circuits by available resources inside FPGA, and put the circuits in suitable place in FPGA chip, and connect between these circuits according to circuit design, and connect pins of FPGA chip with circuit parts inside the chip.
6. Generator of the bit stream, the generator of the bit stream or program file that contain on all information about circuit design and comparison the design by available resources inside FPGA, and how connect the switches inside FPGA. [7]

1.3.12 FPGA Comparison

Historically, FPGAs have been slower, less energy efficient and generally achieved less functionality than their fixed ASIC counterparts. A combination of volume, fabrication improvements, research and development, and the I/O capabilities of new supercomputers have largely closed the performance gap between ASICs and FPGAs

Advantages include a short time to market, ability to re-program in the field to fix bugs, and lower non-recurring engineering costs.

- IC costs are rising aggressively
- ASIC complexity has bolstered development time and costs
- Revenue losses for slow time to market are increasing
- Financial constraints in a poor economy are driving low cost technologies

These trends make FPGAs better than ASICs.

In addition, some FPGAs have the cability of partial re-configuration that lets one portion of the device be re-programmed while other portion continue runing. [3]

1.4 What is Difference Between FPGA and CPLD?

Both are programmable digital logic chips and are made by the same companies. But they have difference characteristics.

1. FPGAs are fine-grain devices that means contain a lot (up to 100000) of tiny blocks of logic with FFs . CPLDs are coarse-grain devices, they contain few (100s max) large blocks of logic with FFs.
2. FPGAs are RAM based they need to be downloaded (configured) at each power up. CPLDs are EEPROM based they are active at power up.
3. FPGAs have special routing resources to implement efficiently arithmetic functions. CPLDs do not have them.
4. CPLD have faster input to output timings than FPGAs.
5. FPGAs can contain large digital designs, while the CPLDs can contain little designs. [6]

1.5 VHDL Language

1.5.1 Definition

VHDL (VHSIC (Very High Speed Integrated Circuit) hardware description language) is hardware description language used in electronic design automation to describe digital and mixed signal system as field programmable gate arrays and integrate circuit. [8]

1.5.2 About VHDL

VHDL was originally developed at the behest of the US Department of Defense in order to document the behavior of the ASICs that supplier companies were including in equipment.VHDL was developed as an alternative to huge, complex manuals which were subject to implementation-specific details, [8] was that development in 1980s.

The initial version of VHDL , designed to IEEE standard 1076 - 1987, the second issue of IEEE 1076, in 1993, some minor change in the standard (2000 and 2002), in Jun 2006, VHDL Technical Committee of Accellera approved so called Draft 3.0 of VHDL 2006, in September 2008 the VHDL standard IEEE 1076 2008 was approved by REVCOM. [8]

1.5.3 Basic Structure of a VHDL file

The VHDL code consists of a design entity can contain other entities also, and an architecture body. The entity declaration the interface to the outside world that defines the input and output signals, the architecture body contains the description of entity and is composed of interconnected entities, processes and components, as figure 1.10. [9]

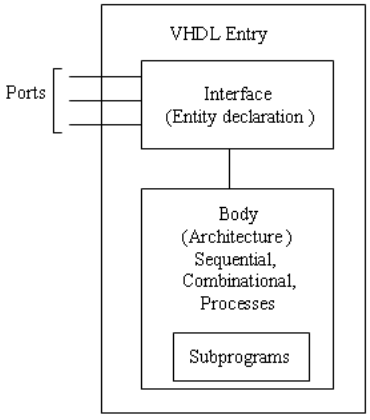


Figure 1.10 VHDL Structure

Entity Declaration

An entity is list a with specifications of all input and output pins (port) of the circuit, as followed code.

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

The mode of the signal IN (unidirectional), OUT (unidirectional), INOUT (bidirectional), or BUFFER. The type of signal can be BIT, STD_LOGIC, INTEGER, etc. The name of entity can be any name except VHDL reserved world. [10]

Architecture

The architecture is a description of how the circuit should behave (function), as following code.

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

An architecture has two parts, declarative part (optional), where the signal and constants are declared, and the code part. The name of entity and architecture can be any name except VHDL reserved world. [10]

1.5.4 Some Simple Code Examples by VHDL Language

In VHDL, a design consists at a minimum of an entity, which describes the interface and an architecture, which contains the actual implementation. In addition, most designs import library modules. Some designs also contain multiple architectures and configurations. [8]
A simple AND gate in VHDL would look something like this:

```
entity ANDGAT is
    Port ( A : in std_logic;
           B : in std_logic;
           C : in std_logic;
           Y : out std_logic);
end ANDGAT;

architecture Behavioral of ANDGAT is

begin

   Y <= A and B AND c ;

end Behavioral;
```

Full adder:

```
entity FULL_ADDER is
    Port ( A : in std_logic;
           B : in std_logic;
           C : in std_logic;
           S : out std_logic;
           Co : out std_logic);
end FULL_ADDER;

architecture Behavioral of FULL_ADDER is

beginN
   S <= A xor B xor C ;
   Co <= (A and C) or (B and A) or (B and C) ;

end Behavioral;
```

D Flip-Flop with asynchronous Reset (Triggered at the rising edge)

```
entity DFF is
    Port ( D : in std_logic;
           CK : in std_logic;
           R : in std_logic;
           Q : out std_logic);
end DFF;

architecture Behavioral of DFF is

begin
  Process (R, CK);
  begin
   IF (R = '1') then
    Q <= '0' ;
    Elsif (CK'event and CK= '1') THEN
     Q <= D ;

        end IF ;
       end Process ;
      end Behavioral ;

  end Behavioral;
```

2. Description of Basic Logic Functions

2.1 Introduction

In this chapter we explain some logic circuits that can be execute by FPGA , actually all the logic circuits can be execute in FPGA approximately , from simple logic circuits to complex logic circuits as microcontroller or microprocessors.

2.2 The Main Logic Gates and other Gates

The main logic gates they are basic for building any logic systems, the logic gates they are work with binary numbers, the voltage that used with logic gates , they are high (logic (1)) or low (logic (0)) . The logic systems builder by use three basic logic gates the are AND gate, OR gate and NOT gate. [11]
The main logic gates and the other gates are shown in table 2.1

A	B	$A \cdot B$	$A+B$	$\overline{A}$	$\overline{A \cdot B}$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
0	0	0	0	1	1	1	0	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1

Table 2.1 The Logic gates

2.3 Full Adder (FA)

The full adder circuit calculate very simple mathematical function, it have three inputs (A,B,C) , and two outputs sum (S) and carry (Co) . The full adder circuit can be add three binary bit , as shown in figure 2.1 and table 2.2. [12]

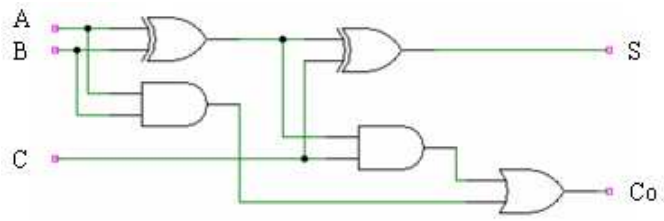


Figure 2.1 Full adder circuit

A	B	C	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 2.2 Truth table of full adder

2.4 Digital Comparison

The Comparison logic circuit able to comparison the binary numbers, as shown in figure 2.2 this circuit able to comparison two digital numbers , each number consists on four bits A (A0, A1, A2, A3) and B (B0, B1, B2, B3), and determine only if the numbers equal or no. the output of the circuit (Y) , when Y = high (logic (1)) then the numbers are equal , when Y = low (logic (0)) then the numbers are different .[13]

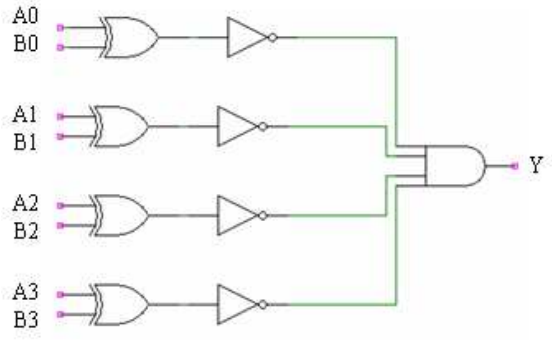


Figure 2.2 4 Bit comparator circuit

But multi-bit comparator can be constructed to compare whole binary or BCD words to produce an output if one words is larger, equal to, or less than the other, as shown in figure 2.3 4 bit magnitude comparator circuit. [14]

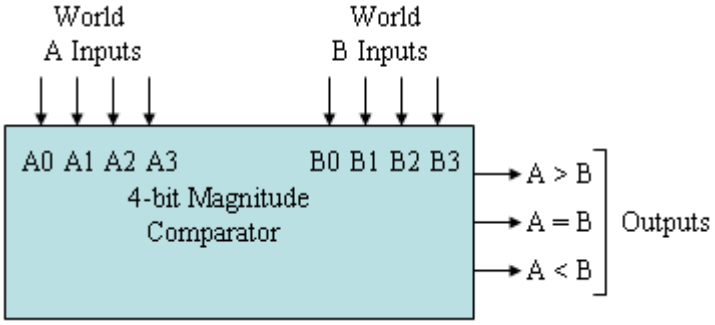


Figure 2.3 4 Bit magnitude comparator circuit

2.5 Shift Register

In Figure 2.4 Shift Register Circuit serial input and serial output, the data it will be shifting on position to right side with each rising clock signal.

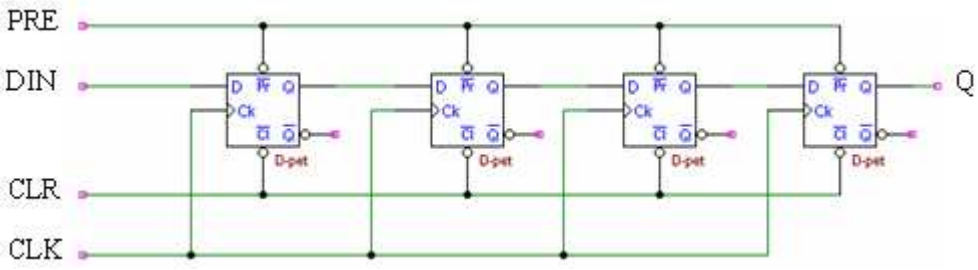


Figure 2.4 Shift Register circuit

2.6 Multiplexers

A multiplexer (MUX) or data selector can be used to select one of four operations as shown in figure 2.5. The output Q of multiplexer. The input functions A,B,C,D (the input functions for example Adder, Subtract, Multiplier,…, any function) . The C1,and C2 are the selection inputs of the multiplexer.

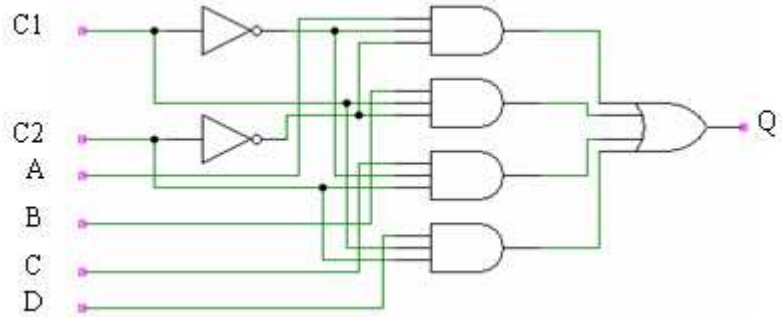


Figure 2.5 Multiplexer circuit [15]

2.7 Full subtracter

The logic circuit for full subtracter shown in figure 2.6 , where the minuend = A , the subtracted = B , the internal borrowing = Bin , the difference = Di , the external borrowing = Bo [11]

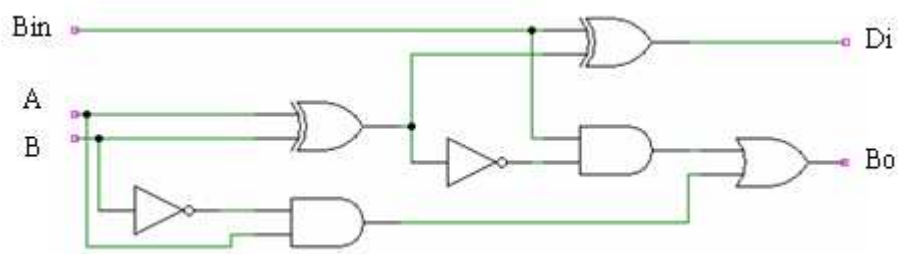


Figure 2.6 Full Subtracter circuit

2.8 The Flip-Flops

2.8.1 JK Flip-Flop

In figure 2.7 the symbol of JK Flip-Flop (Positive Edge Triggered). it has J , K inputs , and clock input , preset input active at logic (0), and clear input active also at logic (0), as table 2.3, the Q(n) output of flip-flop before input clock signal (or before change clock signal) , Q(n+1) output of flip-flop after clock signal input (or after change clock signal) .[12]

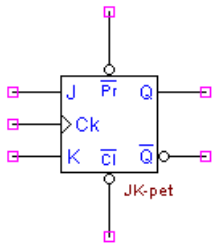


Figure 2.7 JK Flip-Flop

J	K	Q(n)	Q(n+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table 2.3 Truth table JK Flip-Flop

2.8.2 D Flip-Flop

The figure 2.8 it shown the symbol of D flip-flop (Positive Triggered). It has data input (D), clock input (CK), set input (PR), clear input, and output (Q). If the input D at the state logic (1) before input clock signal then the next state after input clock signal it will be at logic (1), if the D at the stat (0) before input clock signal then the next state after input clock signal it will be logic (0) , as shown in truth table 2.4 .[12]

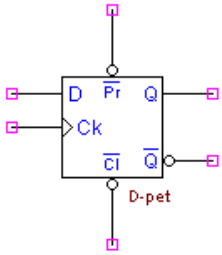


Figure 2.8 D Flip-Flop

D	Q(n)	Q(n+1)
0	0	0
0	1	0
1	0	1
1	1	1

Table 2.4 Truth table for D Flip-Flop

2.9 Decoder

Decoder from binary code to decimal (0 - 7)

The Decoders that shown in figure 2.15 , it will be conversion the binary code (A B C) to decimal number (from 0 to 7) , the input for this decoder binary code , the output for this decoder 8 LED when the binary code is congruent to the decimal number the LED for this number it will be light . As Figure 2.9. [13]

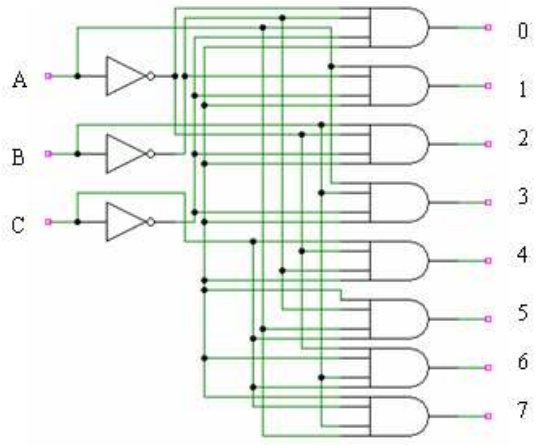


Figure 2.9 Decoder circuit

2.10 Synchronous Counter

In Figure 2.10 synchronous counter by used D Flip-Flop, with count enable signal (E), preset signal (PRE), clear signal (CLR), the counter it will be count from (0000) at (1111). [16]

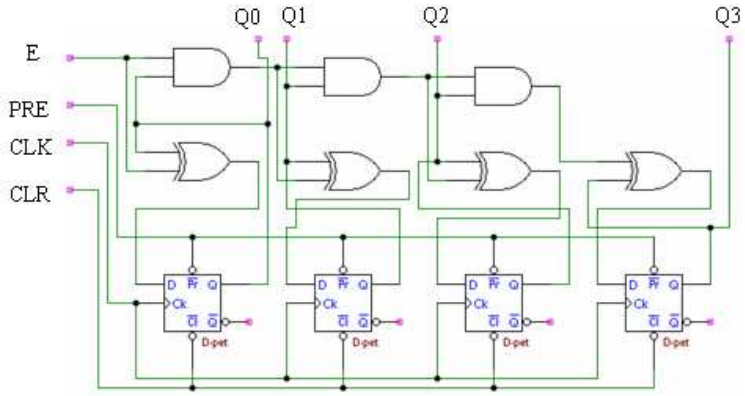


Figure 2.10 Binary counter circuit

2.11 Traffic Lights Control

The Traffic lights that used in cross ways, for regular the moving the cars and the pedestrian, the mode of the traffic lights are some different between the different the countries, but in generally the traffic lights have for cars green color, red color, and orange or yellow color, for pedestrian green color, and red color. The green light allows traffic to proceed in direction denoted, the red light prohibits any traffic from proceeding, the orange or yellow light denoting prepare to stop or going but for cars only. Will be designed using FSM and then translated into VHDL, for that a structural description will be used.

2.12 Binary Multiplier

The circuit of binary multiplier is shown in figure 2.11 , for multiplier two binary numbers , each binary number consists of two bits (A0 , A1) , (B0 , B1) normally the output four bits (Q0 , Q1 , Q2 , Q3).

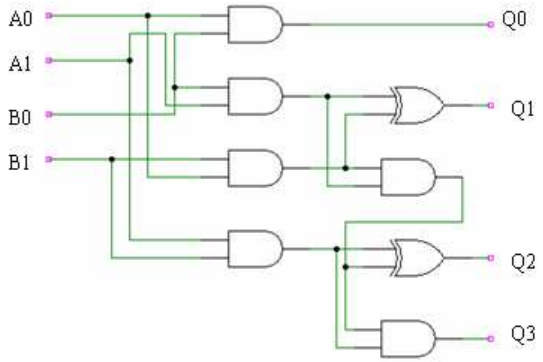


Figure 2.11 Binary multiplier circuit

2.13 Memory (RAM)

2.13.1 Block Ram

In Figure 2.12 shown Memory RAM 16x4 for write or read on it. It can be storage 16 world , each world consists on 4 bits , the total bits that storage in this memory 64 bits , it consists input data (D0 D1 D2 D3) , input adders (A0 A1 A2 A3) , output data (O0 O1 O2 O3) , and WE input for select the write or read mode. [11]

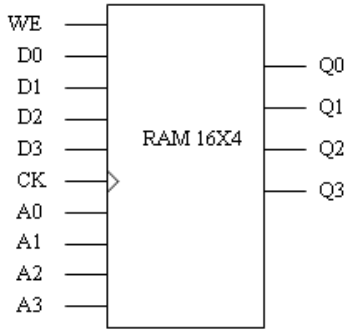


Figure 2.12 RAM 16x4 bits

2.13.2 Distributed RAM

In each FPGA specifically SPARTAN- 3 Configurable Logic Block (CLB) contains up 32 bits of dual-port RAM. This RAM is distributed throughout the FPGA and is commonly called "distributed RAM" to distinguish it from block RAM. [17]

2.14 Debounce circuit

The Debonce circuit it be used with mechanical switches(like limit switches and push buttons) for provide the logic circuits by clean pulse, as figure 2.13, rather than used mechanical switch alone, that give several signals when move this switch from close to open or from open to close.

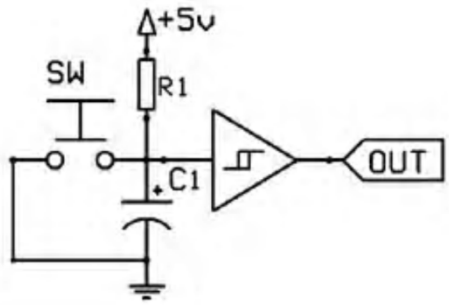


Figure 2.13 Debonce circuit [18]

2.15 Priority Encoder Circuit

The priority encoder circuit used where need change the data bits , for example from 4 bits input to 2 bits output , or from 8 bits input to 3 bits output , or from 16 bits input to 4 bits output , shown in figure 2.14 priority encoder 8-to-3 bit priority encoder , and truth table for this encoder as table 2.5 . [19]

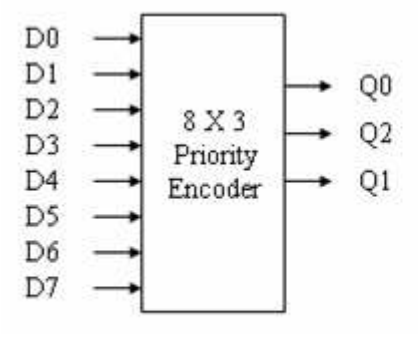


Figure 2.14 8-to-3 Priority Encoder

D7	D6	D5	D4	D3	D2	D1	D0	Q2	Q1	Q0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

Table 2.5 Truth table for Priority Encoder

2.16 Digital Stopwatch

In figure 2.15 Digital stopwatch (00 – 99 sec) that relies used four Integrate circuits

1. 1 x CD4060BM (14 stage ripple carry binary counter)
2. 1 x CD4040BM (14 stage ripple carry binary counter)
3. 1 x MC14518B (BCD counter) D. 2 x MC14511B (BCD to seven segment)
4. 2 x 7 segment LED displays driver [20]

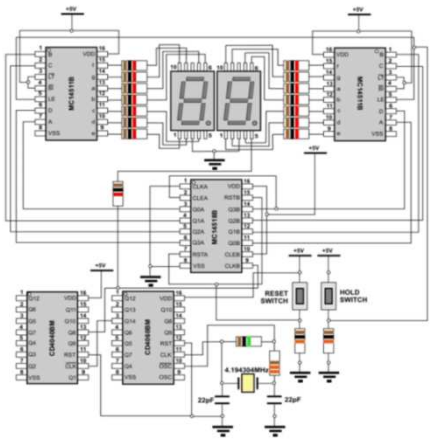


Figure 2.15 Digital stopwatch [20]

2.17 Barrel Shifter

The diagram of barrel shifter is shown in figure 2.16. The input for this circuit is 8 bits , the output is shifted version of the input with the amount of shift defined by shift input (from 0 to 7) . The circuit consists of three individual barrel shifters, the first barrel has only one '0' connected to one of the multiplexers, while the second has two, and the third has four. [10]

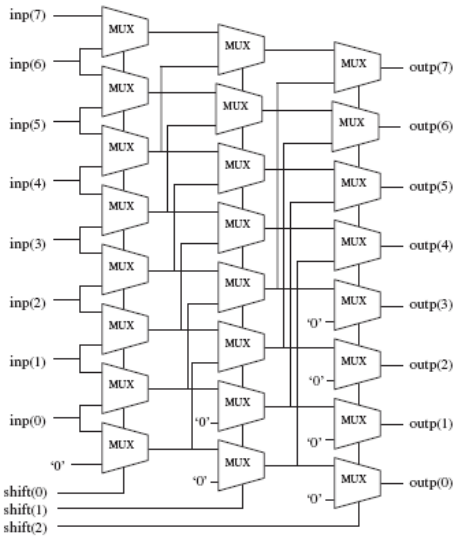


Figure 2.16 Barrel shifter [10]

2.18 LD Latch

LD is a transparent data latch. The data output (Q) of the latch reflects the data input (D), while the gate enable (G) input is High. The data on the (D) input during the High-to-low gate transition is stored in the latch. The data on the (Q) output remains unchanged as long as gate (G) remains Low, as figure 2.14 and table 2.6. [21]

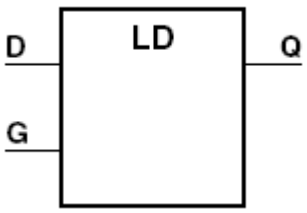


Figure 2.17 LD Latch

G	D	Q
1	0	0
1	1	1
0	X	No change
↓	D	D

Table 2.6 Truth table LD Latch

2.19 Double Data Register (DDR Register)

DDR Register is a dual data rate (DDR) output D flip-flop with clock enable (CE) and asynchronous preset (PRE) and clear (CLR). It consists of one output buffer and one dual data rate flip-flop (FDDRCPE), as figure 2.18, and trout table 2.7.

When the asynchronous PRE is High and CLR is Low, the Q output is preset High. When CLR is High, Q is set Low. Data on the D0 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C0 clock transition. Data on the D1 input is loaded into the flip-flop when PRE and CLR are Low and CE is High on the Low-to-High C1 clock transition. [22]

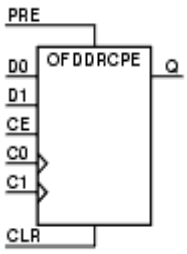


Figure 2.18 DDR Register [22]

Inputs							Outputs
C0	C1	CE	D0	D1	CLR	PRE	Q
X	X	X	X	X	1	0	0
X	X	X	X	X	0	1	1
X	X	X	X	X	1	1	0
X	X	0	X	X	0	0	No Chg
↑	X	1	D0	X	0	0	D0
X	↑	1	X	D1	0	0	D1

Table 2.7 Truth table DDR Register

2.20 Digital Accumulator

An accumulator is build with an adder whose sum can be loaded into a register as shown in figure 2.19. Accumulators are a basic building block of most large digital logic [23]

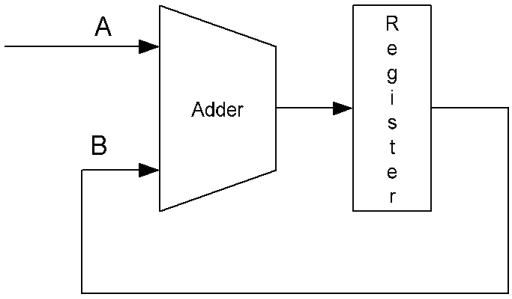


Figure 2.19 Digital accumulator

3. Design of Logic Emulator Structure

3.1 Introduction

In this chapter, execute several logic circuits by VHDL Language and testing these logic circuits practically inside FPGA chip.

3.2 The Tools that used for Execute Logic Circuits

Computer

1. For writ, the logic circuits by VHDL Language and download these logic circuits that written by FHDL Language inside FPGA Chip via XILINX software.
2. FPGA Chip type Spartan 3
3. Adapter

For supplied FPGA Chip by DC power

4. Conductor cable

For connected between Computer and FPGA Chip

3.3 Spartan 3 FPGA Board

The Spartan 3 FPGA board is a robust board containing many features. A list of key features and their location on the board is listed below. As shown in figure 3.1

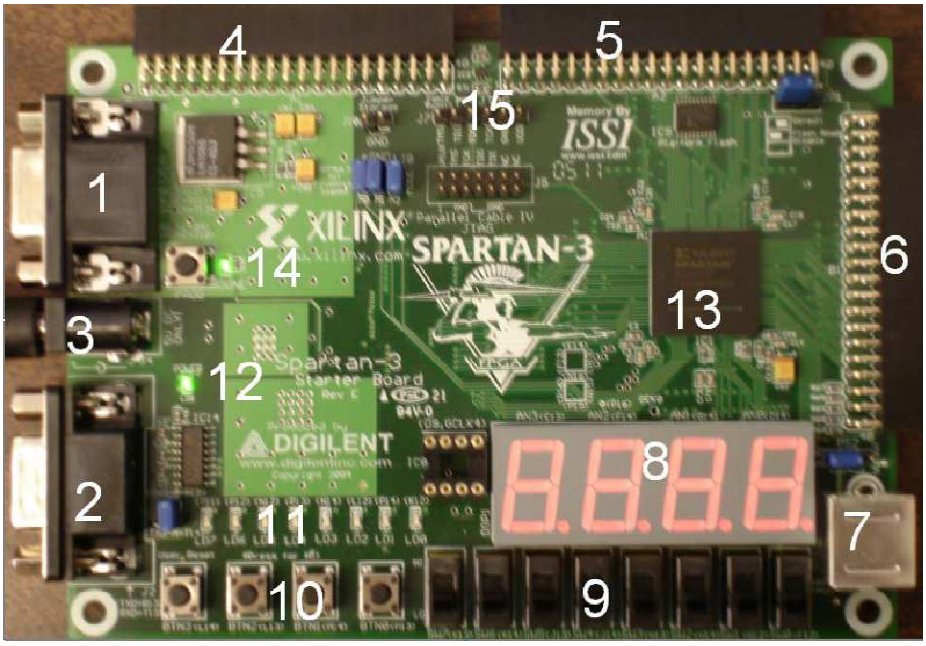


Figure 3.1 Spartan-3 FPGA Board [24]

1. VGA (HD-15) Monitor Port
2. 9-pin (DB-9)
3. Power Connector
4. A1 Expansion Port
5. A2 Expansion Port
6. B1 Expansion Port
7. PS/2 Port
8. Seven Segment Displays
9. Switches (8)
10. Buttons (4)
11. LEDs (8)
12. Power LED
13. Spartan 3 FPGA Core
14. Program LED (Lit when the FPGA is programmed)
15. JTAG Port (used to program The FPGA)

Will notice that next to each component on the board, a label and pin number is printed on the board. For example, the leftmost button has "BTN3 (L14)" printed on it, the "L14" is the pin number we will use in Xilinx. There are also 3, 40-pin expansion ports along the edges of the board that we can use for wiring the FPGA to external devices. Note that not all of these pins can be used. [24]

3.4 Diagram of Digital Circuit Emulator with Programmable Logic

In figure 3.2 shown the diagram of FPGA chip. It contain on several functions (from function 1 at n for example). These functions that designed and written by VHDL code, and downloaded inside FPGA chip. Contain also on multiplexer for connected the selected function by output, clock divider for divided signal clock, counter function for select function , and seven segment driver. The outside FPGA chip data input (4 push buttons , 8 switches), output (8 LEDs, 4 seven segment) , and digital clock for generator signal clock.

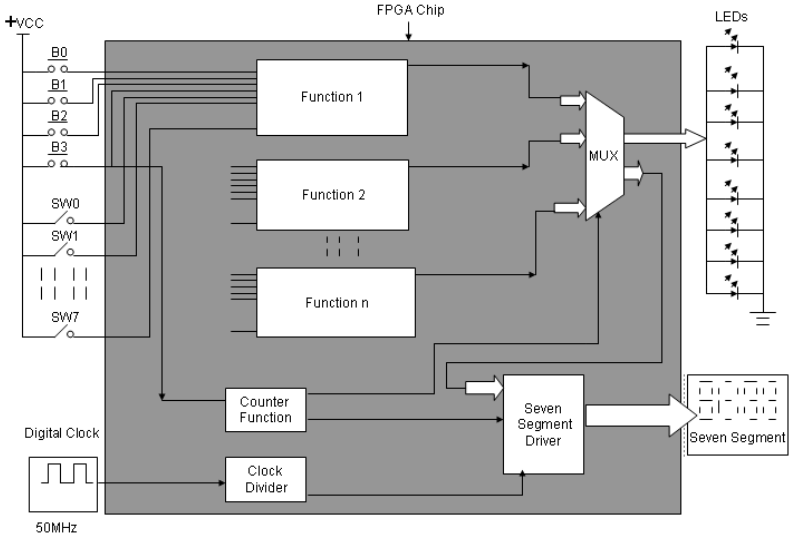


Figure 3.2 Diagram of digital circuit emulator with programmable logic

3.5 Design of Some Logic Functions by VHDL Language

3.5.1 Simple Logic Gates

In figure 3.3 simple gates that use inputs , SW7, SW6, SW5, and clock signal , outputs , LED7, LED6, LED5, LED3, LED2, LED1, as following form

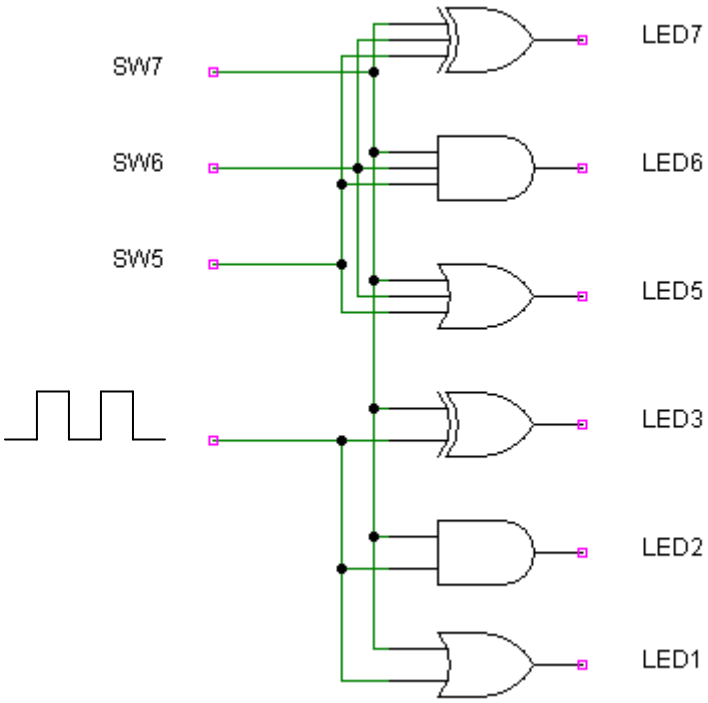


Figure 3.3 Simple gates

Implementation this function by FHDL code

```
236   -- XXXXXXXXXXXXXXX FUNCTION 1 - Simple gates
237     f_LED(1)(7) <= SW(7) xor SW(6) xor SW(5);
238     f_LED(1)(6) <= SW(7) and SW(6) and SW(5);
239     f_LED(1)(5) <= SW(7) or  SW(6) or  SW(5);
240     f_LED(1)(4) <= quad_count(1);
241     f_LED(1)(3) <= SW(7) xor quad_count(1);
242     f_LED(1)(2) <= SW(7) and quad_count(1);
243     f_LED(1)(1) <= SW(7) or  quad_count(1);
244     f_LED(1)(0) <= quad_count(1);
```

3.5.2 Multiplexers Circuit

The Multiplexer Circuit useful in chose the required function rather than use another circuit for generator this function , the inputs for multiplier circuit (SW4 , SW5 , SW6 , SW7 inputs functions , SW1 , SW2 select inputs) , the output LED 7 as figure 3.4 .

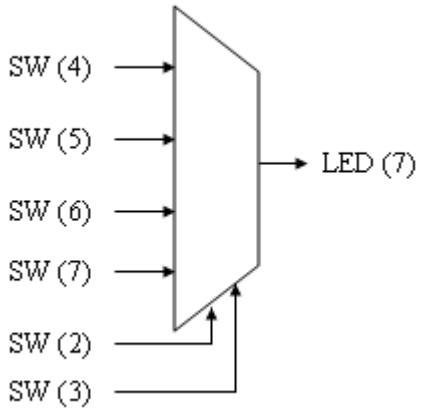


Figure 3.4 Multiplexer circuit

Implementation Multiplexer circuit by VHDL code

The describe of previous multiplexer circuit by VHDL language as following describe the functions inputs SW(7), SW(6), SW(5), SW(4), the select input SW(3), SW(2), the function output selected LED(7), the inputs and outputs are defined in beginning the VHDL design in entity section.

```
247    -- XXXXXXXXXXXXXX FUNCTION 2 - Mux
248      f_LED(2)(7) <= SW(7) when SW(3 downto 2) = "11" else
249                              SW(6) when SW(3 downto 2) = "10" else
250                              SW(5) when SW(3 downto 2) = "01" else
251                              SW(4) ;
```

3.5.3 Decoder Circuit

This Decoder Circuit for decoder the binary numbers (000 till 111) to decimal numbers (0 till 7) , the inputs for this circuit are (Din SW7 , SW6 , SW5 , SW4 , clear input SW4) , the outputs are (LED7 .. LED0) , each LED to mean on decimal number , for example the LED0 to mean on number 0 in decimal , the LED7 to mean on number 7 in decimal system , as figure 3.5.

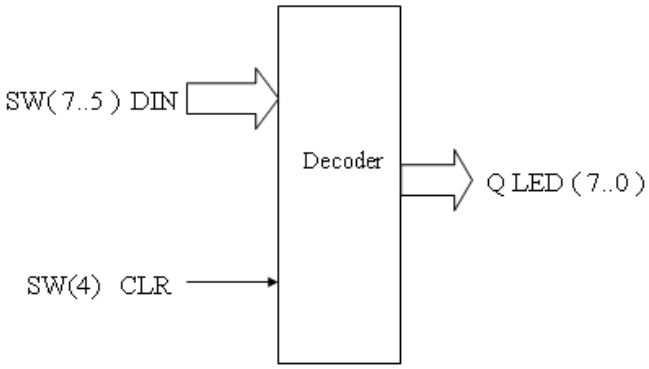


Figure 3.5 Decoder

Implementation the Decoder circuit by VHDL code

This code VHDL language for decoder tree binary bit to decimal (from 0 to 7) , the inputs for this decoder (SW7 , SW6 , SW5) , and outputs (LED0 , to LED6) on the FPGA board , the inputs and outputs are defined in beginning the VHDL design in entity section.

```
265   -- XXXXXXXXXXXXXX FUNCTION 3 - Decoder
266   -- SW(7..5) - DIN
267   -- SW(4) -  CLR
268   -- LED(7..0)
269     process(SW(7 downto 4))
270     begin
271         if ( SW(4) = '0') then
272            f_LED(3) <= "00000000";
273         else
274            case SW(7 downto 5) is
275               when "000" => f_LED(3) <= "00000001";
276               when "001" => f_LED(3) <= "00000010";
277               when "010" => f_LED(3) <= "00000100";
278               when "011" => f_LED(3) <= "00001000";
279               when "100" => f_LED(3) <= "00010000";
280               when "101" => f_LED(3) <= "00100000";
281               when "110" => f_LED(3) <= "01000000";
282               when "111" => f_LED(3) <= "10000000";
283               when others => f_LED(3) <= "00000000";
284            end case;
285         end if;
286     end process;
```

3.5.4 Binary Multiplier Circuit

The circuit of binary multiplier is shown in figure 3.6 , for multiplier two binary number , each binary number consists of four bits.

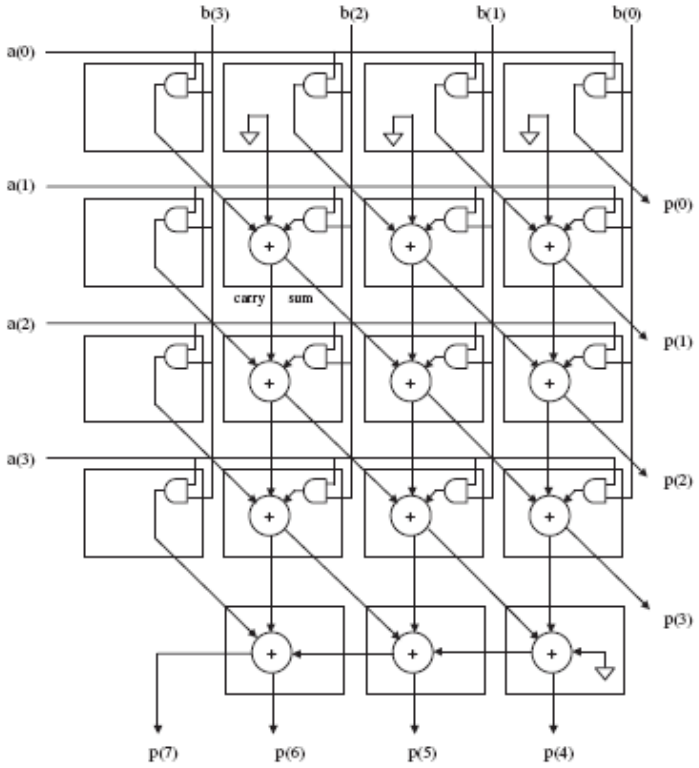


Figure 3.6 Binary Multiplier circuit [10]

Implantation the multiplier circuit by VHDL language

This code VHDL language for multiplier circuit for multiplier two binary number each number consisting on four bits , the inputs for this multiplier for first number (SW7, SW6, SW5, SW4) and for second number (SW3, SW2, SW1, SW0), and output for this multiplier seven segment display , these inputs and outputs are defined in beginning VHDL design in entity section

```
634             when 3 => DIN(15 downto 8) <= SW(7 downto 4) * SW(3 downto 0);
```

3.5.5 D Flip-Flop Circuit

The circuit of D Flip-Flop shown in figure 3.7 , the inputs are (Din SW (3..0) , clear BTN (3) , clock BTN (2) , clock enable SW (7)) , the outputs are (Q LED (3..0)) , the data input will be go to output with each rising edge of clock input.

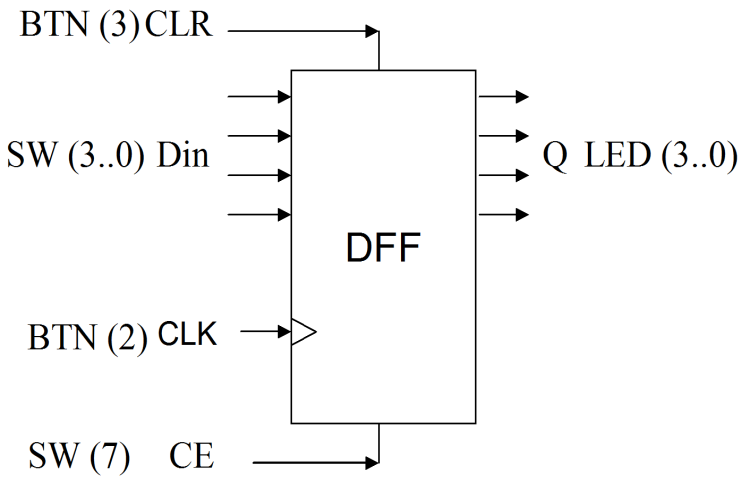


Figure 3.7 D Flip-Flop Circuit

Implementation the D Flip-Flop Circuit by VHDL language
The input for this D Flip-Flop(SW3 , SW2 , SW1 , SW0) , the output (LED3, LED2, LED1, LED0), clock input BTN(2), reset BTN(3), enable SW(7), these inputs and outputs are defined in beginning VHDL design in entity section.

```
351   -- XXXXXXXXXXXXXX FUNCTION 8 - FF, FF_1
352   -- BTN(3) - CLR
353   -- BTN(2) - CLK
354   -- SW(7)  - CE
355   -- SW(3..0) - DIN
356   -- LED(3..0)- Q
357
358     process (BTN(3), BTN(2))
359     begin
360       if BTN(3) = '1' then
361             f_LED(8)(3 downto 0)      <= (others => '0');
362         elsif BTN(2)'event and BTN(2) = '1' then          -- Rising edge
363             if SW(7) = '1' then
364                   f_LED(8)(3 downto 0) <= SW(3 downto 0);
365             end if;
366         end if;
367     end process;
```

3.5.6 Synchronous Counter Circuit

In figure 3.8 4 bits Synchronous Counter Circuit by use DFF , the inputs are (clock enable SW (7) , clock SW (2) , clear BTN (3)), the outputs are (Q LED (3..0)) , as VHDL code below

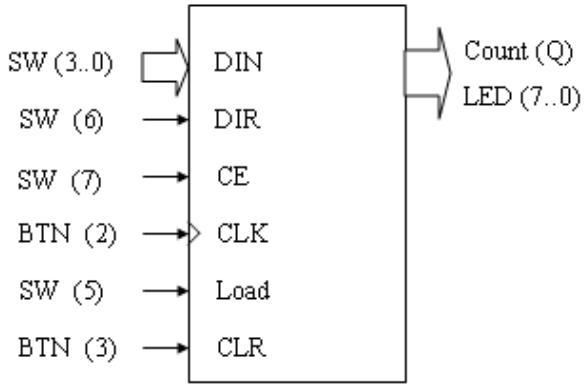


Figure 3.8 Synchronous Counter Circuit by use DFF

Implementation the 4 bits Synchronous Counter Circuit by VHDL language

```
415   -- XXXXXXXXXXXXXX FUNCTION 11 - COUNTER
416
417   -- BTN(3)- CLR
418   -- BTN(2)- CLK
419   -- SW(7) - CE
420   -- SW(6) - DIR
421   -- SW(5) - LOAD
422   -- LED(7..0)- Q
423   -- SW(3..0)- DIN
424        process (BTN(3), BTN(2))
425        begin
426        if BTN(3)= '1' then  -- Active high reset
427         count <=(others => '0');
428        elsif BTN(2)'event and BTN(2) = '1' then  -- Rising edge
429         if SW(7)= '1' then -- CE
430          if SW(5) = '1' then  -- LOAD
431           count(3 downto 0) <= SW(3 downto 0);
432          else
433           if SW(6) = '1' then   -- DIR
434              count <= count + 1 ;
435            else
436              count <= count - 1 ;
437            end if;
438          end if;
439         end if ;
440        end if ;
441        end process ;
442
443         f_LED(11) <= count;
```

3.5.7 Debounce Circuit

In figure 3.9 Debounce Circuit that use buffer , resistor and capacitor , the input for this circuit (SW) , the output (OUT) for clean pulse.

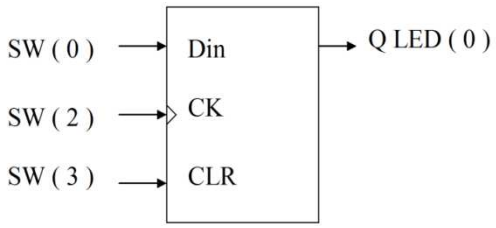


Figure 3.9 Debounce Circuit

Implementation Debounce Circuit by VHDL language

```
510   -- XXXXXXXXXXXXXX FUNCTION 14 - Debounce circuit
511   -- SW(0) - DIN
512   -- BTN(3) - CLR
513   -- BTN(2) - CLK
514   -- LED(0) - Q
515
516
517    process(BTN(2), BTN(3))
518   begin
519     if (BTN(3) = '1') then
520       f_LED(14)(1) <= '0';
521       f_LED(14)(2) <= '0';
522       f_LED(14)(3) <= '0';
523     elsif (BTN(2)'event and BTN(2) = '1') then
524       f_LED(14)(1) <= SW(0);
525       f_LED(14)(2) <= f_LED(14)(1) ;
526       f_LED(14)(3) <=f_LED(14)(2) ;
527     end if;
528   end process;
529
530   f_LED(14)(0) <= f_LED(14)(1) and f_LED(14)(2) and (not f_LED(14)(3) );
531
```

3.5.8 Shift Register Circuit

The Shift Register Circuit that shown in figure 3.10 that parallel load , and parallel output , the inputs (Din SW (3..0) , clock enable SW (7) , clock BTN (2) , load SW (5) , shift lift logic SW (6) , clear BTN (3)) , the output (Q LED (7..0)) , as VHDL code below

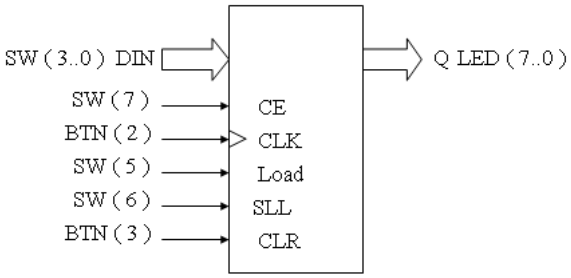


Figure 3.10 Shift Register

Implementation the Shift Register by VHDL code

```
510   -- XXXXXXXXXXXXXX FUNCTION 14 - Debounce circuit
511   -- SW(0) - DIN
512   -- BTN(3) - CLR
513   -- BTN(2) - CLK
514   -- LED(0) - Q
515
516
517    process(BTN(2), BTN(3))
518   begin
519     if (BTN(3) = '1') then
520       f_LED(14)(1) <= '0';
521       f_LED(14)(2) <= '0';
522       f_LED(14)(3) <= '0';
523     elsif (BTN(2)'event and BTN(2) = '1') then
524       f_LED(14)(1) <= SW(0);
525       f_LED(14)(2) <= f_LED(14)(1) ;
526       f_LED(14)(3) <=f_LED(14)(2) ;
527     end if;
528   end process;
529
530   f_LED(14)(0) <= f_LED(14)(1) and f_LED(14)(2) and (not f_LED(14)(3) );
531
```

3.5.9 Digital Comparator Circuit

In figure 3.11 4 bit comparator circuit for comparator two binary numbers , and determine if the numbers are equal or which number is greater than or which number is less than, it have data inputs number A (SW7 , SW6 , SW5 , SW4) , number B (SW3 , SW2 , SW1 , SW0) , outputs LED7 (A > B) , LED6 (A = B) , LED5 (A < B).

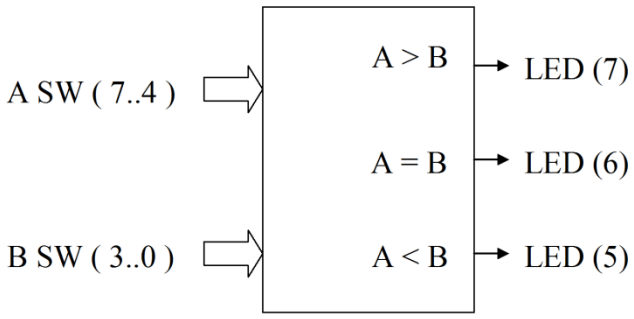


Figure 3.11 4-bit Comparator Circuit

Implementation the 4 bit Comparator Circuit by VHDL code

```
337   -- XXXXXXXXXXXXXX FUNCTION 7 - Comparator
338   -- SW(7..4) = NUMBER A
339   -- SW(3..0) = NUMBER B
340   -- LED(7) -- A > B
341   -- LED(6) -- A = B
342   -- LED(5) -- A < B
343
344   f_LED(7)(7) <= '1' when SW(7 downto 4) > SW(3 downto 0) else -- GT (Greater Than)
345                        '0';
346   f_LED(7)(6) <= '1' when SW(7 downto 4) = SW(3 downto 0) else -- EQU (EQUal to)
347                        '0';
348   f_LED(7)(5) <= '1' when SW(7 downto 4) < SW(3 downto 0) else  -- LT (Less Than)
349                        '0';
```

3.5.10 Digital Accumulator

The Digital Accumulator Circuit , building of adder and register as figure 3.12 , the inputs are (Din SW(3..0) , clock signal BTN (2)) , the outputs are (Q LED (3..0)) , as VHDL code below

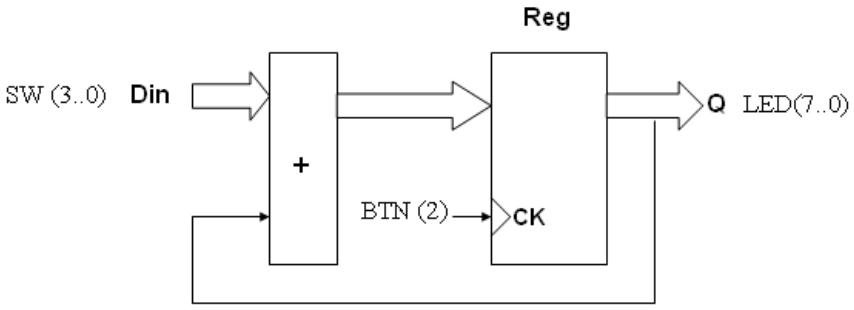


Figure 3.12 Digital Accumulator

Implementation the Digital Accumulator Circuit by VHDL Code

```
302   -- XXXXXXXXXXXXXX FUNCTION 5 - Accumulator
303
304   -- BTN(3) - CLR
305   -- BTN(2) - CLK
306   -- SW(7) - CE
307   -- SW(3..0) - DIN
308   -- LED(7..0)- Q
309      process (BTN(3), BTN(2))
310        begin
311        if BTN(3) = '1' then
312         f_LED(5)(7 downto 0) <= (others => '0');
313        elsif BTN(2)'event and BTN(2) = '1' then  -- Rising edeg
314         if SW(7) = '1' then
315          f_LED(5)(7 downto 0) <= f_LED(5)(7 downto 0) + sw(3 downto 0);
316         end if;
317        end if;
318        end process;
319
```

3.5.11 Latch Circuit

In figure 3.13 shown the LD Latch circuit, the inputs for this circuit are (D data input SW (1), Gate SW (0)), the output (Q LED (0)), the data on input D is transit from input to output when the gate enable (G = 1), while the data stored when the gate (G = 0), as VHDL code below.

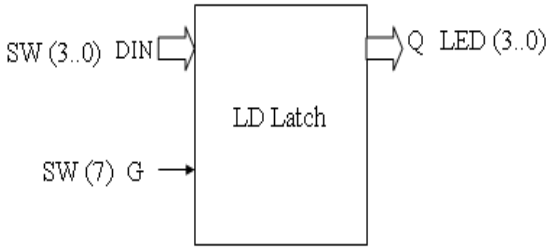


Figure 3.13 LD Latch

Implementation the Latch Circuit by VHDL code

```
379   -- XXXXXXXXXXXXXXX FUNCTION 9 - Latch
380   -- SW(7)-- GATE
381   -- SW(3..0)-- DIN
382   -- Q -- LED(3..0)
383
384    process (SW(7), SW(3 downto 0))
385   begin
386      if SW(7)='1' then  --GATE active High
387         f_LED(9)(3 downto 0) <= SW(3 downto 0);
388      end if;
389   end process;
```

3.5.12 Priority Encoder Circuit

The Priority Encoder Circuit must encode the address of the input bit from 8 bits to 3 bits , the priority encoder circuit use in several applications such as keyboard . The inputs for this circuit are (SW(7..0)) , and outputs are (LED(2..0)), as shown in figure 3.14.

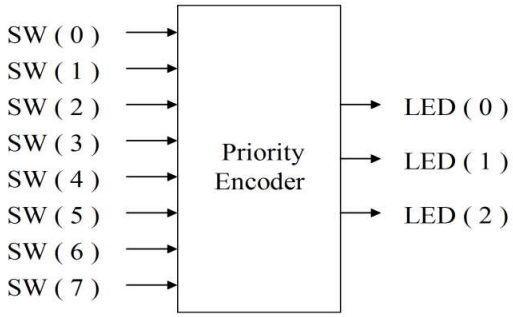


Figure 3.14 Priority Encode

Implementation the Priority Encoder by VHDL code

```
288   -- XXXXXXXXXXXXXX FUNCTION 4 - priority encoder
289   -- SW(7..0) - DIN
290   -- LED(2..0) - Q
291
292      f_LED(4)(2 downto 0) <= "000" when SW(0) = '1' else
293     "001" when SW(1) = '1' else
294     "010" when SW(2) = '1' else
295     "011" when SW(3) = '1' else
296     "100" when SW(4) = '1' else
297     "101" when SW(5) = '1' else
298     "110" when SW(6) = '1' else
299     "111"  ;
```

3.5.13 Barrel Shifter Circuit

Figure 3.15 shows the four bits Barrel Shifter Circuit, the circuit shift the input vector (of size 4) either 0 or 1 position to the lift. for example when the input equal (0011), and shift equal (00), the output it will be equal to input (0011), when the shift (01), the output it will be (0110), when the shift (10), the output it will be (1100), when the shift (11), the output it will be (1001), and so on.

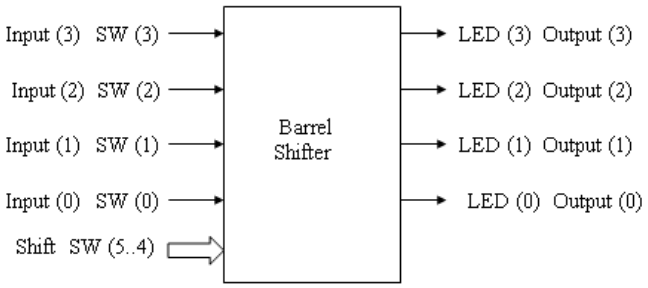


Figure 3.15 Barrel Shifter

Implementation the Barrel Shifter Circuit by VHDL code

```
321          -- XXXXXXXXXXXXXX FUNCTION 6 - Barrel Shifter
322         -- SW(5..4)- SHIFT
323         -- SW(3..0)- DIN
324         -- LED(3..0)
325          process (SW(5 downto 4),SW(3 downto 0))
326           begin case SW(5 downto 4) is
327            when "00" =>        -- Shift by 0
328             f_LED(7)(3 downto 0) <= SW(3 downto 0);
329            when "01" =>        -- Shift by 1
330            f_LED(7)(1) <= SW(0);
331            f_LED(7)(2) <= SW(1);
332            f_LED(7)(3) <= SW(2);
333            f_LED(7)(0) <= SW(3);
334            when "10" =>        -- Shift by 2
335            f_LED(7)(2) <= SW(0);
336            f_LED(7)(3) <= SW(1);
337            f_LED(7)(0) <= SW(2);
338            f_LED(7)(1) <= SW(3);
```

```
339            when "11" =>        -- Shift by 3
340            f_LED(7)(3) <= SW(0);
341            f_LED(7)(0) <= SW(1);
342            f_LED(7)(1) <= SW(2);
343            f_LED(7)(2) <= SW(3);
344            when others =>
345            f_LED(7)(3 downto 0) <= SW(3 downto 0);
346          end case;
347        end process;
```

3.5.14 Stop Watch Timer Circuit

Figure 3.16 illustrate the Stop-Watch Circuit that contains on counter and register , that contains on free-running counter, when the signal stop is occur , the status the counter must be stored in the register just before the reset occurs Once stop returns to '0' , the counter resumes counting (from zero) , while the register holds the previous count

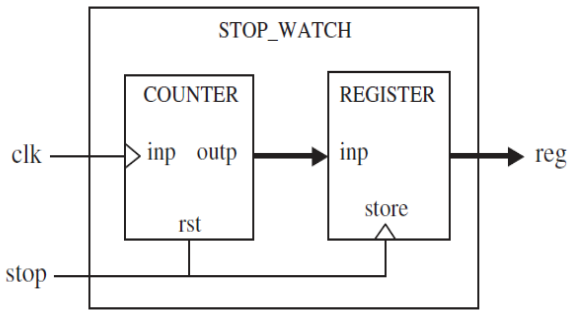


Figure 3.16 Stopwatch Circuit [10]

Implementation the Stop-Watch Circuit by VHDL code

```
591        -- XXXXXXXXXXXXXX FUNCTION 16 - Stop watch timer
592        -- BTN(3)    -  RESET
593        -- BTN(2)    -  START
594        -- BTN(1)    -  STOP
595
596        -- Stop-watch control
597        process (CLK50MHZ)
598        begin
599        if CLK50MHZ'event and CLK50MHZ = '1' then
600         if BTN(1) = '1' or BTN(0) = '1' then  -- STOP button
601          stopw_run <= '0';
602         elsif BTN(2) = '1' then   -- RUN button
603          stopw_run <= '1';
604         end if;
605        end if ;
```

```
606         end process ;
607
608
609         -- 100 Miliseconds counter
610         process (CLK50MHZ)
611         begin
612         if CLK50MHZ'event and CLK50MHZ = '1' then
613          if BTN(3) = '1' then    -- RESET
614           ms100_count <= (others => '0') ;
615          elsif tenhz_en = '1' and stopw_run = '1' then
616               if ms100_count < 9 then
617                 ms100_count <= ms100_count + 1 ;
618               else
619                 ms100_count <= (others => '0') ;
620               end if;
621          end if ;
622         end if ;
623         end process ;
624
625         CE1s <= '1' when ms100_count = 9 and tenhz_en = '1' else
626                 '0';
627
628         -- Seconds counter
629         process (CLK50MHZ)
630         begin
631         if CLK50MHZ'event and CLK50MHZ = '1' then
632          if BTN(3) = '1' then    -- RESET
633           sec_count <= (others => '0') ;
634          elsif CE1s = '1' and stopw_run = '1' then
635               if sec_count < 9 then
636                 sec_count <= sec_count + 1 ;
637               else
638                 sec_count <= (others => '0') ;
639               end if;
640          end if ;
641         end if ;
642         end process ;
643
```

3.5.15 Traffic Lights Control Circuit

In figure 3.17 illustrate the traffic lights, intersection a main road (road of cars has tow directions) with a walker crossing, is controlled by tow traffic lights. The traffic lights for cars has 3 possible value (uppercase): red R, green G, yellow Y, the walker has 2 possible value (lowercase): red r, and green g. There are input signal A (walker push button) to controller that has 2 possible value, 1 (pressed), and 0 (un-pressed). The state machine of traffic lights for this case shown in figure 3.18, that consists on six stages (G r, Y r, R r, R g, R r, R Y r,). The first stage G r, it repeat it self when the signal A = 0, but the state machine will be go to next stage Y r, when signal A = 1 (when, the push button is pressed), after that the state machine continue to next stage,

until first stage G r, with unimportant whether the push button is pressed or no. The figure 3.19 is shown the illumination for this traffic light system for cars and walker.

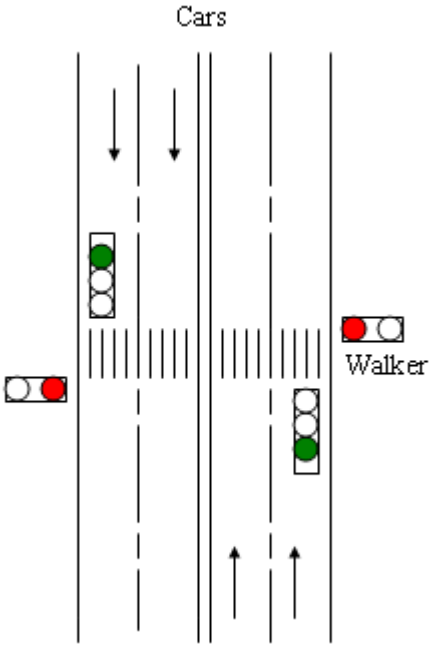


Figure 3.17 Traffic Lights

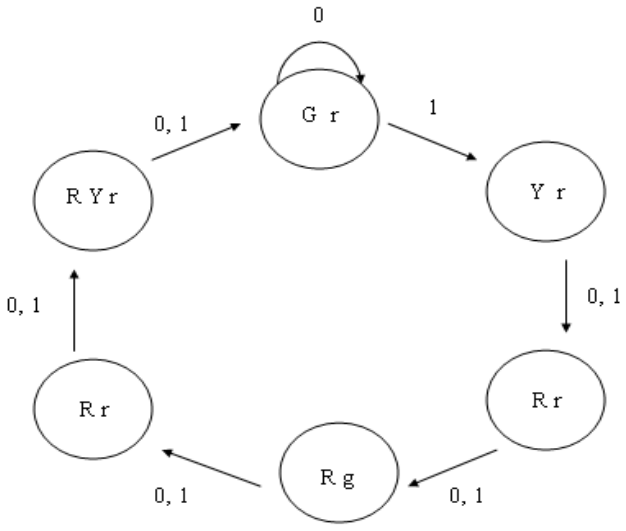
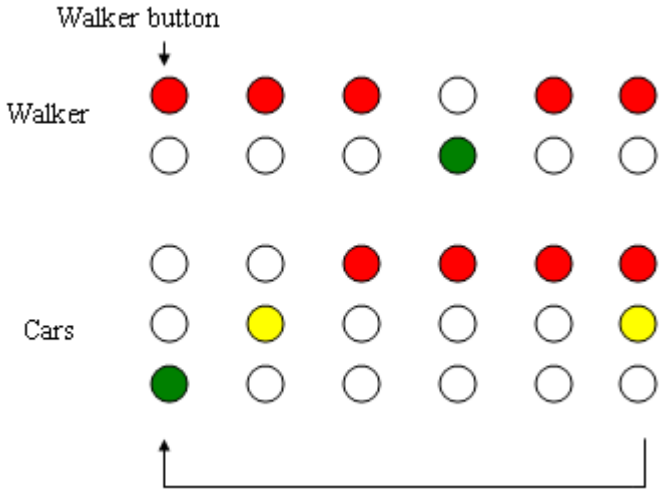


Figure 3.18 Simple Traffic Lights state machine diagram



.

Figure 3.19 Illumination of traffic light control

Connected the Traffic Lights Control with VGA Display

The Spartan-3 board include a VGA display port and DB15 connector, as figure 3.1. Connect this port directly to PC monitor or flat-panel LCD using a standard monitor cable. As shown in figure 3.20 the Spartan-3 FPGA controls by five VGA signals: Red (R), Green (G), Blue (B), Horsetail Sync (HS), and Vertical Sync (VS), each color line has a series of resistors to provide 3-bit color, with one bit each for Red, Green, and Blue. The HS and VS signals are TTL level. Drive the R, G, and B signals High (1) or Low (0) to generate the eight possible colors that shown in Table 3.1.[25] As shown in figure 3.21 Cathode Ray Tube (CRT) display, the VGA controller generate the HS and VS timing signals and coordinates the delivery of video data on each pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the refresh frequency of the display. The number of horizontal lines displayed at given refresh frequency defines the horizontal retrace frequency.[25]

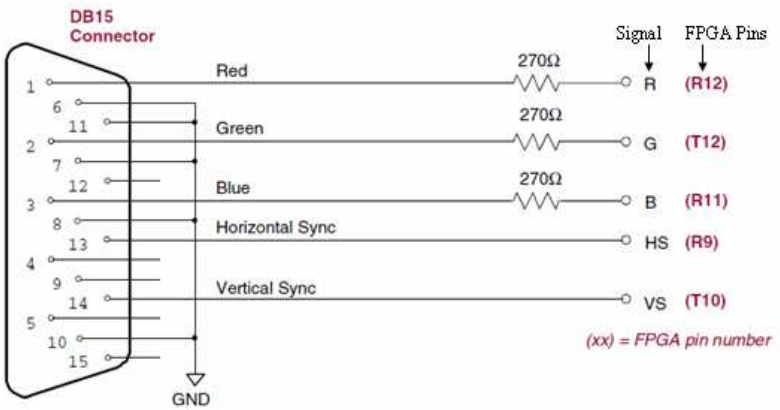


Figure 3.20 VGA Connectors from Spartan-3 Kit board [25]

Red (R)	Green (G)	Blue (B)	Resulting Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Table 3.1 3 Bit display color codes [25]

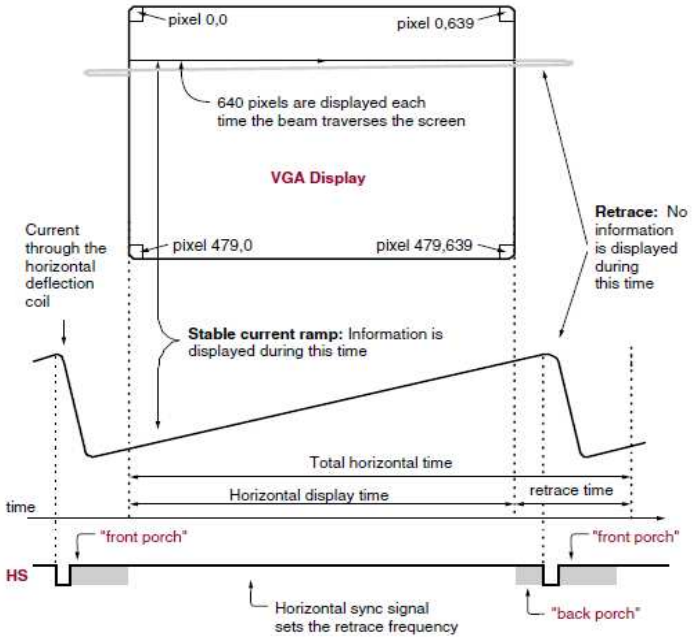


Figure 3.21 CRT Display [25]

The FPGA board will be connected with VGA Display, for display the output of traffic lights for walkers and cars on VGA Monitor, as shown in figure 3.22 output of traffic lights control on VGA display. The graphics features of the implemented functions are powered by a VGA Time-Base for FPGA described in. [26]

Implementation of FSM for traffic lights with VGA display by VHDL code

```
548         -- XXXXXXXXXXXXXX FUNCTION 15 - Traffic lights
549
550         -- BTN(3)    -  RESET
551         -- BTN(1)    -  WALK Request
552         -- LED(7)    -  road RED
553         -- LED(6)    -  road YELLOW
554         -- LED(5)    -  road GREEN
```

```
555         -- LED(4)    -  walker RED
556         -- LED(3)    -  walker GREEN
557         -- LED(2)    -  FSM(2)
558         -- LED(1)    -  FSM(1)
559         -- LED(0)    -  FSM(0)
560
561          trafficFSM: traffic
562           port map(CLK => quad_count(3),
563                 RST    => BTN(3),
564                 WALKRQ => BTN(1),
565                 FSM    => f_LED(15)(2 downto 0),
566                 L2_GREEN  => L2_GREEN ,
567                 L2_RED    => L2_RED   ,
568                 L3_GREEN  => L3_GREEN ,
569                 L3_RED    => L3_RED   ,
570                 L3_YELLOW => L3_YELLOW );
571
572         f_LED(15)(3) <= L2_GREEN;
573         f_LED(15)(4) <= L2_RED;
574         f_LED(15)(5) <= L3_GREEN;
575         f_LED(15)(7) <= L3_RED;
576         f_LED(15)(6) <= L3_YELLOW;
577
578          P3_RED     <= '1' when VGA_X = "010000----" and VGA_Y = "000100----" else
579                    '0';
580          P3_YELLOW  <= '1' when VGA_X = "010000----" and VGA_Y = "000101----" else
581                    '0';
582          P3_GREEN   <= '1' when VGA_X = "010000----" and VGA_Y = "000110----" else
583                    '0';
584          P2_RED     <= '1' when VGA_X = "010010----" and VGA_Y = "000100----" else
585                    '0';
586          P2_GREEN   <= '1' when VGA_X = "010010----" and VGA_Y = "000101----" else
587                    '0';
588          FRAME   <= '1' when (P3_RED or P3_YELLOW or P3_GREEN or P2_RED or P2_GREEN)
         VGA_X = "------0000" or VGA_Y = "------0000") else
589                    '0';
```

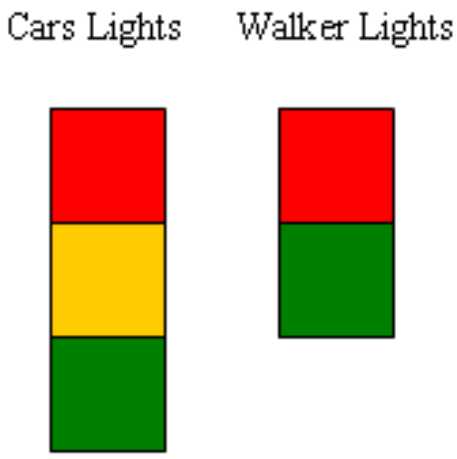


Figure 3.22 Traffic lights output on VGA Monitor

The State Machine of traffic lights Control, as shown in figure 3.23

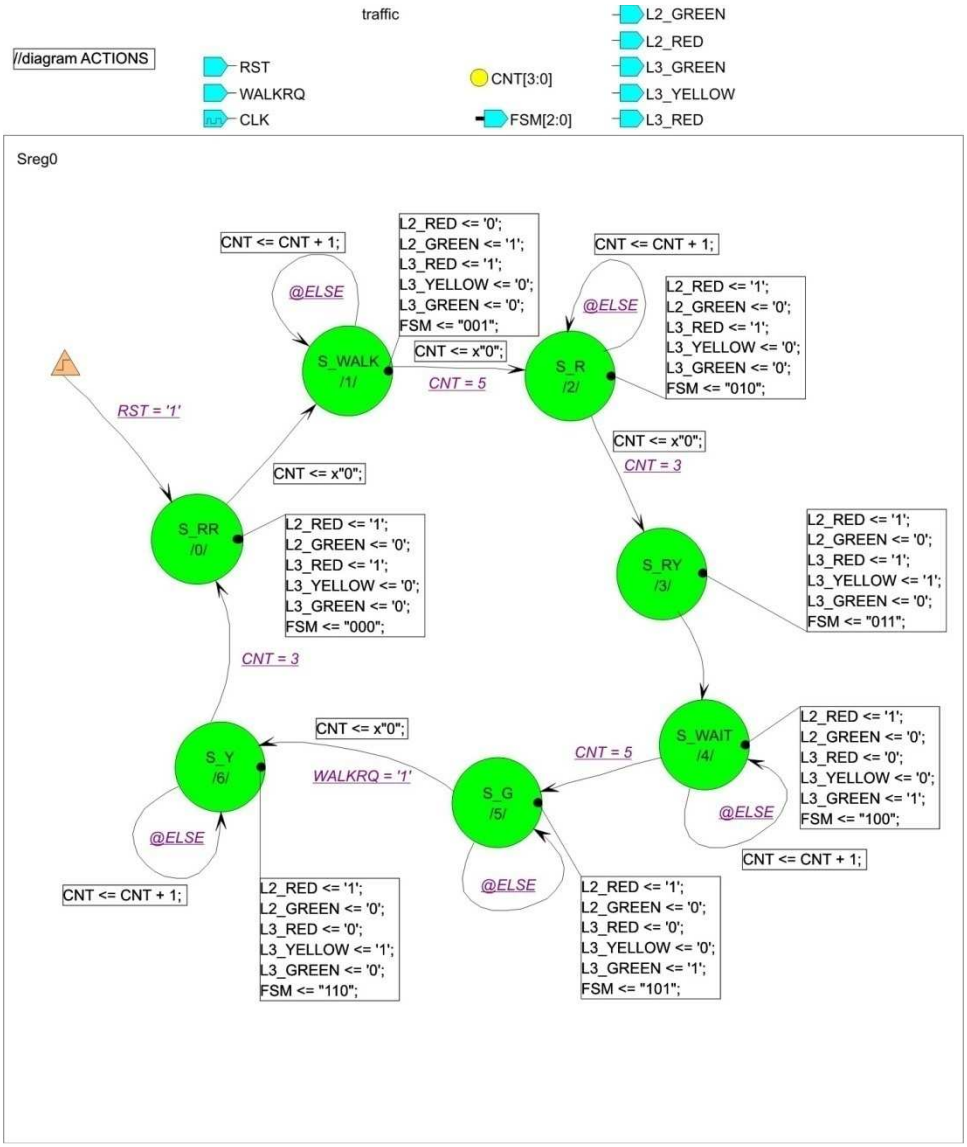


Figure 3.23 FSM for Traffic Lights Control

3.5.16 Double Data Rate Register (DDR Register)

Figure 3.24 illustrate the DDR Register circuit, it contains tow flip-flops, and multiplexer. The FFs connected together to data inputs (DIN), Clock enable (CE) , Clock signal (CLK) , and Clear (CLR) , but each FF has output , each output for these FF are connected to multiplexer input, the multiplexer has final output (Q). The Data on (DIN) input is loaded into the first FF, (the data will be transfer from FF input to FF output) with rising edge on (CLK) clock input. The Data also on (DIN) input is loaded into the second FF with falling edge on (CLK) Clock input.

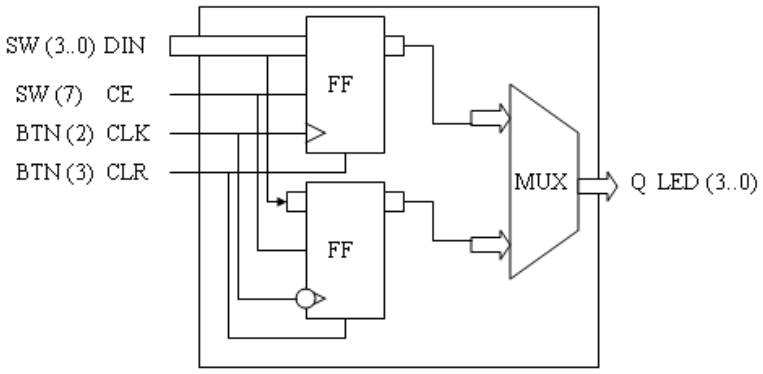


Figure 3.24 DDR Register

Implementation the DDR Register by VHDL code

```
533   -- XXXXXXXXXXXXXX FUNCTION 13 - DDR Register
534   -- BTN(3)- CLR
535   -- BTN(2)- CLK
536   -- SW(7) - CE
537   -- SW(3..0) - DIN
538   -- LED(3..0)- Q
539     f_LED(13)(3 downto 0) <= f_LED(8)(3 downto 0) when BTN(2)= '1' else
540                                                    f_LED(8)(7 downto 4);
```

3.5.17 Distributed RAM Memory

In FPGA can use Look-Up Tables as Distributed RAM. Figure 3.25 illustrate Distributed RAM dual- port has one read / write port and an independent read port.

Write Operations:

The write operation is a single clock-edge operation, controlled by the write-enable input, WE. WE is active High. When the write enable is High, the clock edge latches the write address and writes the data on the D input into the selected RAM location.

When the write enable is Low, no data is written into the RAM.

Read Operation:

The address port either for single or dual-port modes is asynchronous with an access time equivalent to a LUT logic delay. [17]

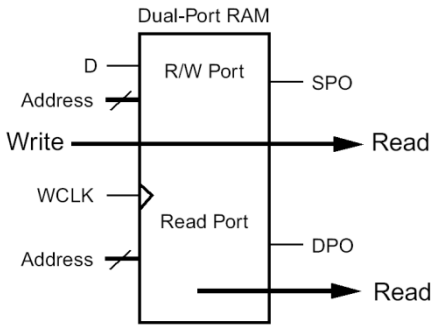


Figure 3.25 Dual-Port Distributed RAM [17]

Connected the Memory with VGA Display

The FPGA board will be connected with VGA Display, for display the output of the memory (addresses according value that stored inside the memory) on VGA Monitor.

Implementation Dual-port Distributed RAM with VGA Display by VHDL code

```
445   -- XXXXXXXXXXXXXX FUNCTION 12 - Memory
446   -- (fcn_index = 12) - WRITE ENABLE
447   -- BTN(2)- CLK
448   -- SW(7 .. 4)- Port A ADDRESS
449   -- SW(3 .. 0)- Port A DATA Input
450   -- LED(3.. 0)- Port A DATA Output
451   ---------------------------------
452   -- RAMADDRB(3..0) - Port B ADDRESS
453   -- RAMDATAB(3..0) - Port B DATA
454
455    process (BTN(2))         -- Port A
456    begin
457      if (BTN(2)'event and BTN(2)= '1') then
458       if (fcn index = X"C") then     -- (fcn index = 12) - WRITE ENABLE
459            RAM(conv_integer(SW(7 downto 4))) <= SW(3 downto 0);
460         end if;
461        end if;
462    end process;
463    f LED(12)(3 downto 0) <= RAM(conv integer(SW(7 downto 4)));
464    RAMDATAB <= RAM(conv_integer(RAMADDRB));
465
466    process (CLK50MHZ) -- Generates 25MHz clock for vga_time
467    begin
468     if CLK50MHZ'event and CLK50MHZ = '1' then
469      CLK25MHZ <= not CLK25MHZ;
470     end if;
471    end process;
472
473    vga1: vga_time
474     port map (CLK => CLK25MHZ,
475                      RESET => BTN(3),
476                      BLANK => open,
477                      DISP => DISP,
478                      DISP_HO -> open,
479                      DISP VO => open,
480                      HS => HS,
481                      LFO => open,
482                      VS => VS,
483                      X => VGA_X,
484                      Y => VGA_Y);
485
486    hexrom: hexchar
487       port map(  HEXVALUE => HEXVALUE,
488                            CHARLINE => CHARLINE,
489                            CHARCOL => CHARCOL,
490                            DOUT => HEXQ);
491
492     CHARLINE <= VGA_Y(3 downto 1);
493     CHARCOL  <= VGA_X(3 downto 1);
494     RAMADDRB <= VGA Y(7 downto 4);
495     HEXBEAM  <= '1' when HEXQ = '1' and VGA_X(9 downto 5) = "00011" and VGA_Y(9 downto 8) =
    "00" else
496                         '0';
497     HEXVALUE <= RAMDATAB when VGA_X(4) = '1' else
498                         RAMADDRB;
499     HEXHILIGHT <= '1' when RAMADDRB = SW(7 downto 4) and VGA_X(9 downto 5) = "00011" and VGA
    Y(9 downto 8) = "00" else
500                               '0';
501
502     VGA_R <= DISP and ( HEXBEAM or (P3_RED and L3_RED) or (P2_RED and L2_RED) or (P3_YELLOW
    and L3_YELLOW));
503     VGA_G <= DISP and ( ( HEXBEAM and not VGA X(4)) or (P3_GREEN and L3_GREEN) or (P2_GREEN
    and L2_GREEN) or (P3_YELLOW and L3_YELLOW));
504     VGA_B <= DISP and ( ( HEXBEAM xor HEXHILIGHT ) or FRAME);
```

4. Implementing the logic design into Spartan-3 Starter Kit Board

4.1 Introduction

In this chapter explain how translate the design of the high level VHDL language, which describe the design to the Register Transfer Level (RTL), into a netlist at the gate level. The second step is optimization, which is performed on the gate level netlist for speed or area, now the design can be simulated, and place-and-rout, software will be generate the physical layout for FPGA chip. [10] As figure 4.1.

The implementation process includes four key steps

1. Translate: Interprets the design and runs (design rule check).
2. Map: Calculates and allocates resources in the targeted device.
3. Place and Rout: Places the CLBs in logical position and utilized the routing resources.
4. Generate Programming File: Creates a programming bitstream. [1]

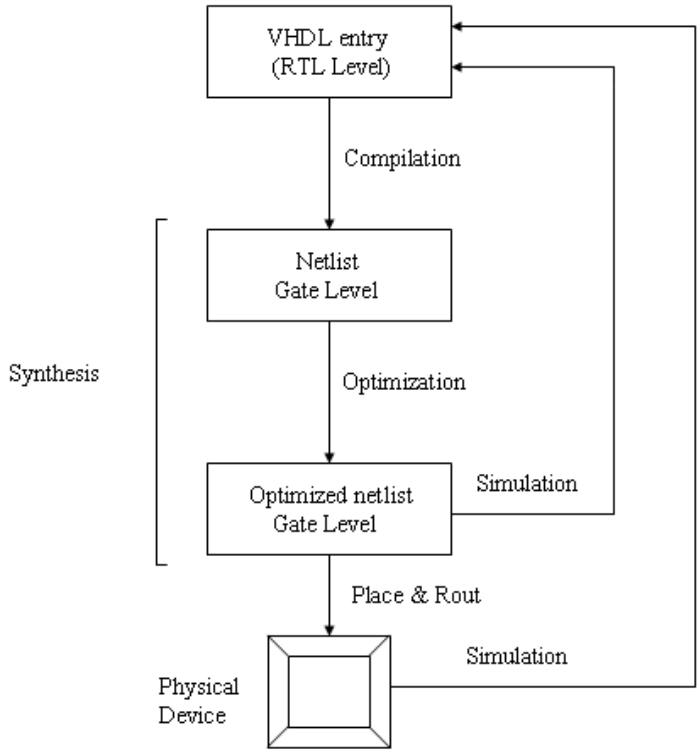


Figure 4.1 VHDL Design flow

4.2 User Constraints

The Constraints Editor is a graphical user interface (GUI) tool for entering timing constraints and pin location constraints. The user interface simplifies constraint entry by guiding you through constraint creation without your needing to understand UCF file syntax. [27]

Assign Package Pins

The Assign Package Pins for few the FPGA Pins specially that connected with the internal logic function inside FPGA chip, double click on Assign Package Pins inside Process Window as figure 4.2 will be obtained on package pin locations as figure 4.3.

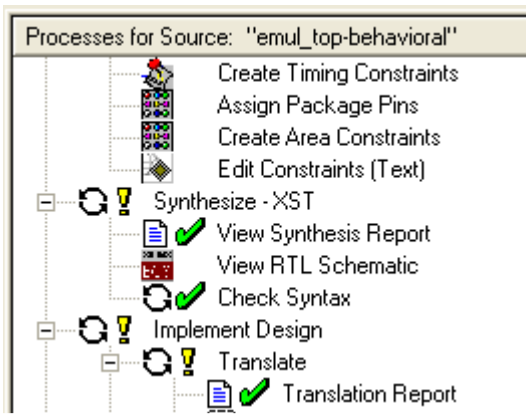


Figure 4.2 Process Window Showing Assign Package Pins

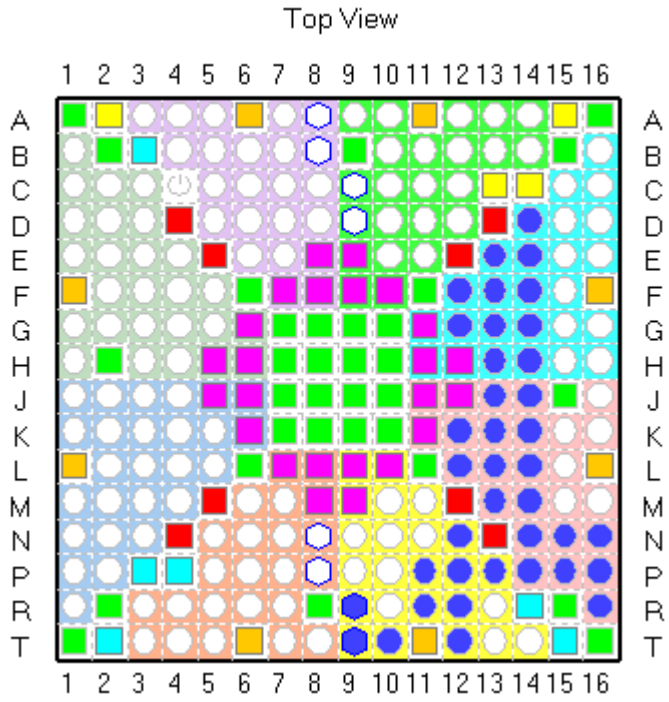


Figure 4.3 Package Pin Locations

4.3 Synthesis

The synthesis stage converts the text-based HDL design into an NGC netlist file. The netlist is a non-readable file that describes the actual circuit to be implemented at a very low level. [1]
In synthesis convert the RTL description to gates , first the RTL description is translate to unoptimized Boolean description (as AND , OR gates , FFs , Latches). Next produce an optimized Boolean equivalent description . Finally , the optimized Boolean equivalent description is mapped to logic gates . [28]

When finish of write the design by VHDL code , check the synthesis by double click on check syntax as figure 4.2 , if the design Ok , the software will show green ckeckmark.

View RTL Schematic

When double click on RTL Schematic in process window as shown in figure 4.2, will be obtained on RTL Schematic of the Logic Emulator Design, as shown in figure 4.4, when click inside the rectangle in figure 4.4 on the software, will obtained on the Detailing RTL Schematic of Logic Emulator Design as shown in the Appendix 1.

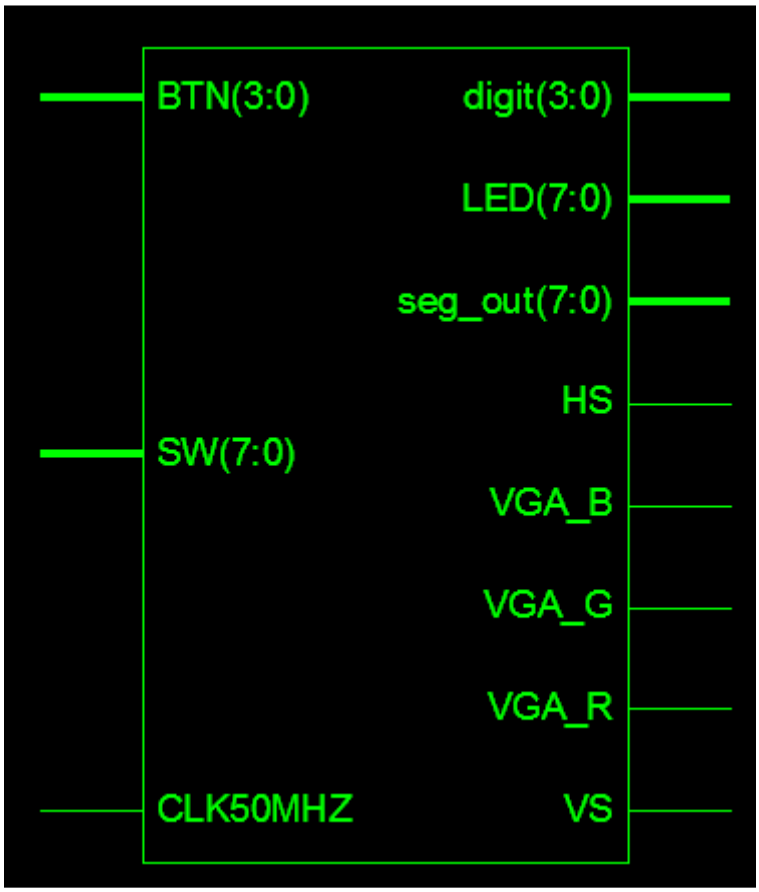


Figure 4.4 RTL Schematic of the Logic Emulator Design

4.4 Implement the Design

4.4.1 Translate

The Translation Stage for translate from RTL description to Boolean equivalent description .

4.4.2 Map

The map stage distributed the design to the resource available in the FPGA . The map stage also uses the UCF file to understand timing. The map has the ability to 'shuffle' the design around LUTs to create the best possible implementation for the design .[1]

Map Report

The Map Report confirms the resources used within the device and describe trimmed and merged logic. And describe where each portion of the design is located in the actual device.[1] Double click on Map Report as figure 4.5 will be obtained Map Report as show in figure 4.6

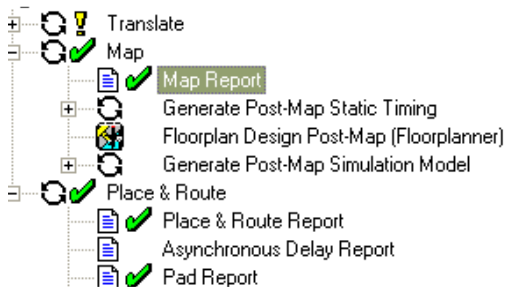


Figure 4.5 Process Window show Implement Design

```
  Total Number Slice Registers:       131 out of   3,840    3%
    Number used as Flip Flops:                123
    Number used as Latches:                     8
  Number of 4 input LUTs:             464 out of   3,840   12%
Logic Distribution:
  Number of occupied Slices:                            274 out of   1,920   14%
    Number of Slices containing only related logic:     274 out of     274  100%
    Number of Slices containing unrelated logic:          0 out of     274    0%
      *See NOTES below for an explanation of the effects of unrelated logic
Total Number of 4 input LUTs:           498 out of   3,840   12%
  Number used as logic:                 464
  Number used as a route-thru:           26
  Number used for Dual Port RAMs:         8
    (Two LUTs used per Dual Port RAM)
  Number of bonded IOBs:                38 out of     173   21%
    IOB Flip Flops:                     11
  Number of MULT18X18s:                 1 out of      12    8%
  Number of GCLKs:                      5 out of       8   62%
```

Figure 4.6 Map Report

Total number slice Register 131 out of 3,840 by rate 3% . Number of total 4 input LUTs 457 out of 3,840 by rate 11% . Number of occupied slices 270 out of 1,920 by rate 14% . Number of bounded IOBs 38 out of 173 by rate 21% . Number of MULTI 18*18s 1 out of 12 by rate 8% . Number of GCLKs 5 out of 8 by rate 62% .

4.4.3 Place & Route

The Place and Rout stage works with allocation CLBs and chose best or suitable location for each block. [1]

The place and route tools read the netlist, extract the components and nets from the netlist, place the components on the target device, and interconnect the components using the specified interconnections. [28]

Place & Route Report

The Place & Rout Report gives a step-by-step progress report.

Double click on Place & Rout Report as shown in figure 4.5 will be obtained on Place & Rout Report as shown in figure 4.7. Number of External IOBs 38 out of 137 by rate 21% . Number of slices 270 out of 1920 by rate 14% . Number of SLICENs 18 out of 960 by rate 1% . Number of BUFGMUXs 4 out of 8 by rate 50% .

```
Device speed data version:  "PRODUCTION 1.39 2007-04-13".


Device Utilization Summary:

   Number of BUFGMUXs                        5 out of 8      62%
   Number of External IOBs                  38 out of 173    21%
      Number of LOCed IOBs                  38 out of 38    100%

   Number of MULT18X18s                      1 out of 12      8%
   Number of Slices                        270 out of 1920   14%
      Number of SLICEMs                     18 out of 960     1%



Overall effort level (-ol):   Standard
Placer effort level (-pl):    High
Placer cost table entry (-t): 1
Router effort level (-rl):    Standard


Starting Placer

Phase 1.1
```

Figure 4.7 Place & Route Report

Floor Planner

The FloorPlanner is a graphical placement tool that gives control over placing design into a target FPGA using a ' drag and drop ' paradigm with the mouse pointer. Double click on FloorPlanner in Process window, as figure 4.5 will be obtained the FloorPlanner as figure 4.8.

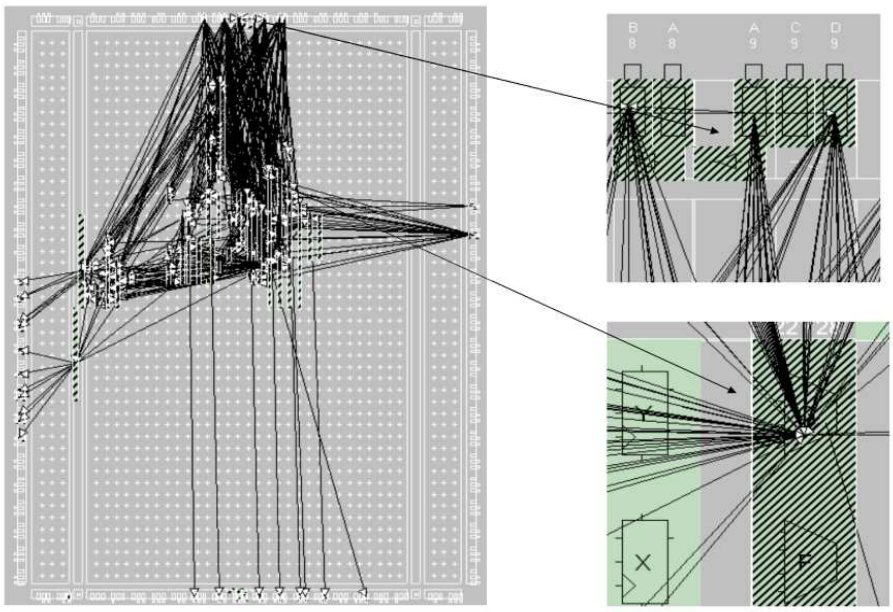


Figure 4.8 FloorPlanner

4.5 Generate Program File

After the Place and Rout Program called 'bitgen' taks the output of the place and rout and creates the program bitstream . The Generate Program File for creates a .bit file that can be used by the iMPACT programmer to configure a device by double click on Generate Program File in process window, as figure 4.9.

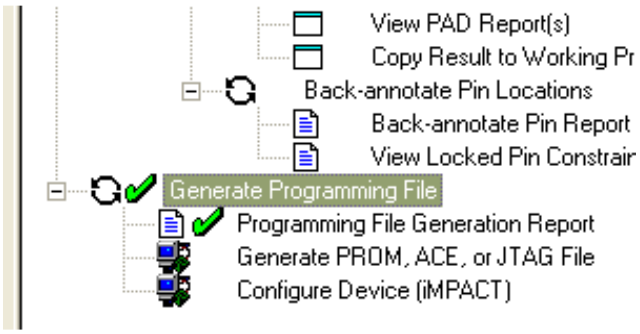


Figure 4.9 Process Window shown Generate Programming File

4.6 The Area Inside FPGA Chip That Used for Implementation The Design

By double click on View / Edit Routed Design will be obtain the area that used for this design as figure 4.10

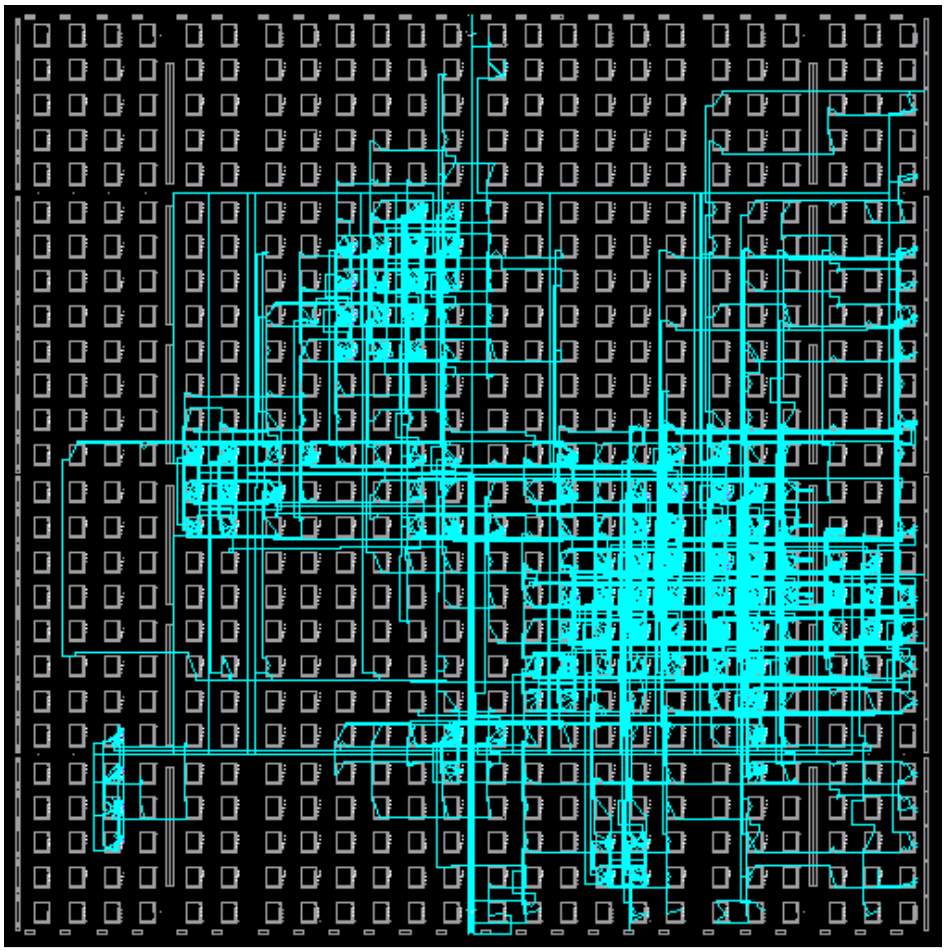


Figure 4.10 The area inside FPGA chip that used for implementation emulator board and logic functions

5. Verification and Testing of the Hardware Emulator

5.1 Introduction

In this Chapter explained verification and tested the some logic functions that designed and written by VHDL language in emulator board.

After download the functions, that written by VHDL code inside FPGA chip. At this point, the FPGA should be programmed and ready, now check and testing some of these functions as verification of this work.

5.1 Testing and Verification of Function No. 11 The Binary Counter Circuit

This circuit for count up and count down by control on count direction , it has clock signal BTN (2) , clock enable signal SW (7) , count direction SW (6) , load signal for control on data input SW(5) , data input SWs (3..0), output LEDs (7..0). This counter will be count up or down according to signal direction with all Rising edge of clock signal, the counter will start the count from 0 from the number that entered on data input, when testing this circuit on FPGA chip , as figure 5.1, obtained on the results as figure 5.2. The function has been successfully tested for several 4 bits DIN.

Figure 5.1 Testing of Binary Counter Circuit (function 11)

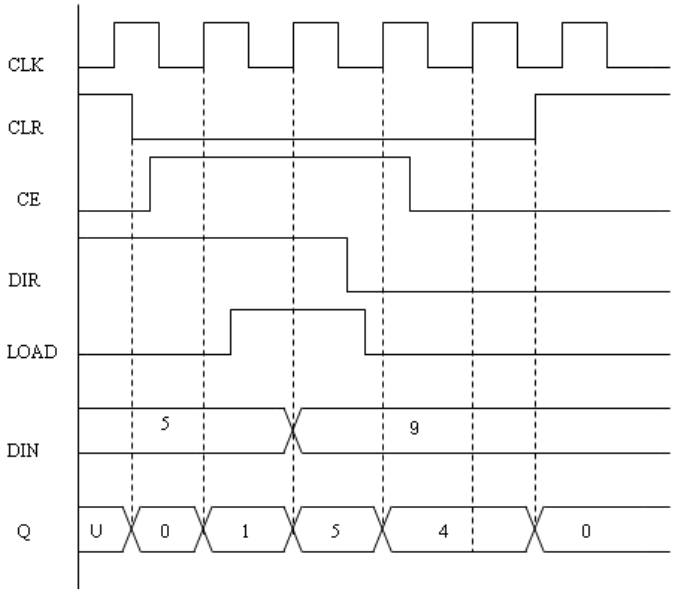


Figure 5.2 Timing Diagram of Binary Counter Circuit (function No. 11)

5.2 Testing and Verification of Function No. 6 The Barrel Shifter Circuit

This Circuit four bits Barrel Shifter, the circuit must be shift the input vector (of size 4). Assumed the input (0011), when the shift equal to (00), the output must be equal to input, when the shift (01), the output must be (0110), and so on, as figure 5.3, when tested and verification the Barrel Shifter Circuit obtained on the required results as timing diagram in figure 2.4. The function has been successfully tested for several 4 bit DIN.

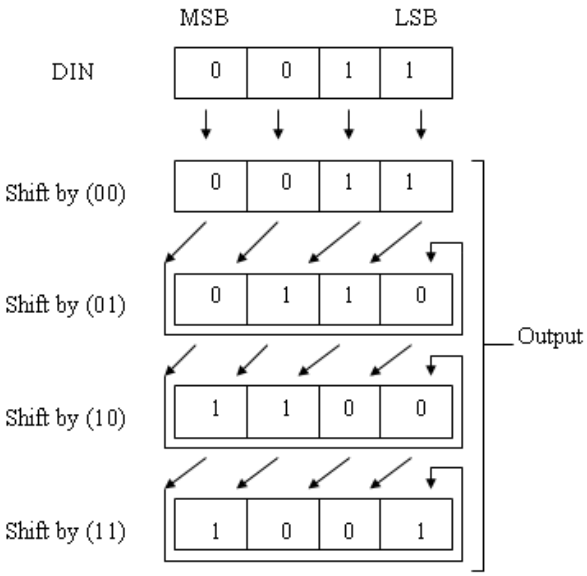


Figure 5.3 Transfer data in 4-bits Barrel Shifter Circuit

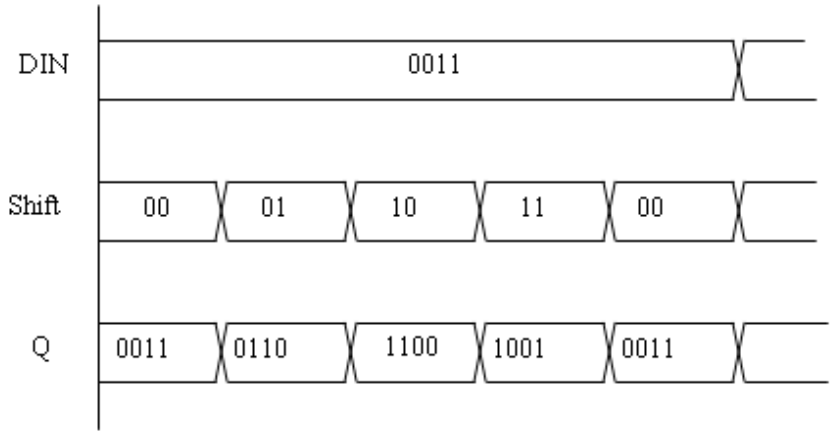


Figure 5 .4 Timing diagram for Barrel Shifter Circuit

5.3 Testing and Verification of Function No. 9 The D Latch Circuit

This circuit for transit the data from input to output, stored data in output, it has data input SWs(3..0), gate SW(7), and data output LEDs(3..0). When the gate enamel (high) the data will be transit to output, the data on the input during change gate from high to low is stored in Latch , the data on the output remains unchanged as long as gate (G) remains, where tested this circuit obtained on results the timing diagram as figure 5.5. The function has been successfully tested for several 4 bit DIN.

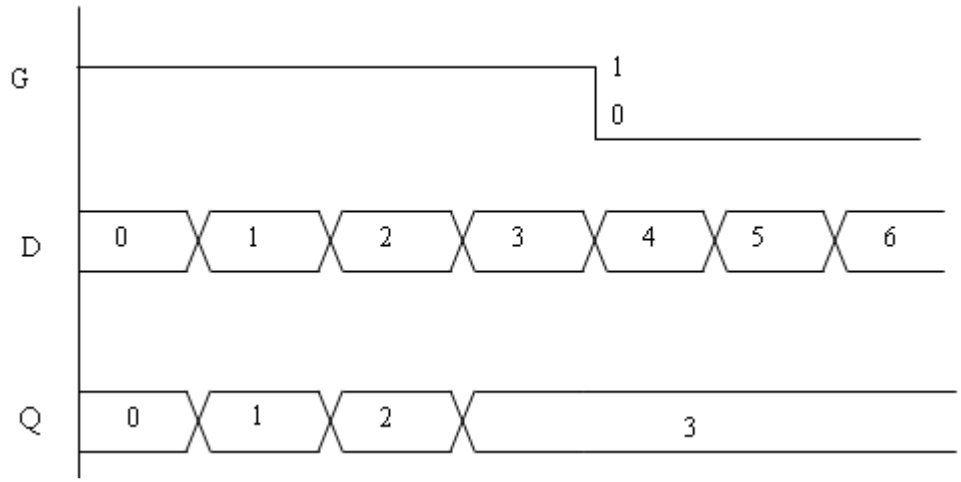


Figure 5.5 Timing diagram of D Latch Circuit

5.4 Testing and Verification of Function No. 3 The Binary Multiplier Circuit

This circuit for multiplier two binary numbers each number consists on four bits, the first number is the SWs (3...0), second number is the SWs (7...4), and output seven segment display. When testing this circuit obtained on results (Timing diagram), as figure 5.6. The function has been successfully tested for several 4 bit DIN.

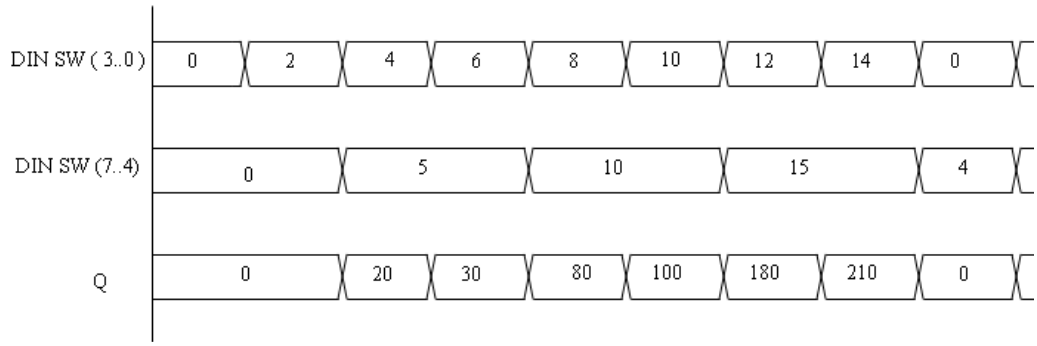


Figure 5.6 Timing Diagram of Multiplier Circuit

5.5 Testing and Verification of Function No. 12 - The Distributed RAM with VGA Display

This function for read and write or store the data inside FPGA chip (dual port, port A, and port B), consists on address input SWs (7..4), data input SWs (3..0), and output LEDs (3..0), this function is connected to VGA connector, for show the memory output (addresses according value that stored inside the memory) on PC screen or LCD. This function shows the output on FPGA board (LEDs), and VGA monitor at the same time. When tested this circuit obtained on results as timing diagram in figure 5.7, and photo of PC screen for VGA display as figure 5.8. The function has been successfully tested for all 16 addresses with several DIN values.

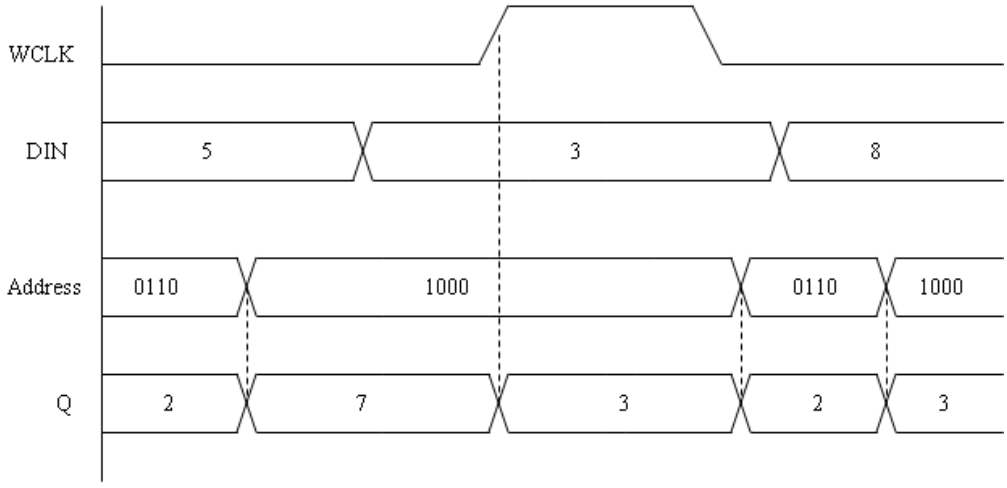


Figure 5.7 Memory RAM Timing diagram

Figure 5.8 Output of RAM memory on VGA Display

5.6 Testing and Verification of Function No. 15 - The Traffic Lights Control with VGA Display

This function consists on inputs, reset signal BTN (3), walker request BTN (1), and outputs, red cars LED(7), yellow cars LED (6), green cars LED(5), red walker LED (4), green walker LED (3), FSM (0) LED (0), FSM (1) LED (1), FSM (2) LED (2). This function will be connected to VGA connector, for show the output traffic lights control on PC screen or LCD. Can show output on FPGA board (LEDs), or on VGA monitor at same time. When testing this function obtained the results on VGA display as shown in figure 5.9. The function has been successfully tested for all states of finite state machine diagram.

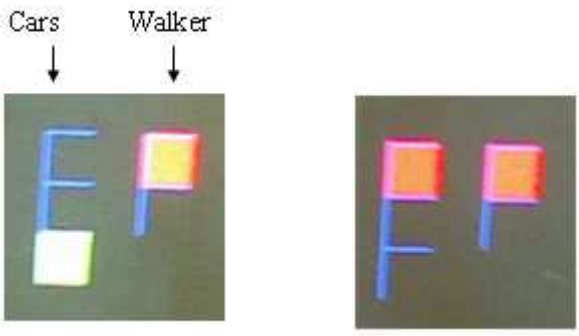


Figure 5.9 Traffic Lights Control output on VGA Display for two states

6. Conclusion

6.1 FPGA Technology Versus Conventional Logic ICs

In this section explain the difference between FPGA Technology and conventional logic, by very simple example (Binary Multiplier Circuit) , this circuit for multiplier two binary number each number consists on tow bit as figure 2.16 , assembling this circuit required unit for testing and assembling this circuit , logic ICs (2 IC 7408 , 1 IC 7486) , external conductors , input (switches) , outputs (LEDs) as figure 6.1 . The assembling this circuit need much more external conductors, time for design and assembling, the PCB size, and the errors possibility. But in FPGA Technology can write similar this circuit by VHDL language in one or tow lines for multiplier several numbers for example tow binary numbers each number consists on four bits as VHDL code in Figure 6.2 . FPGA Technology used one board for execute several logic functions, easier errors detection, short time for design logic functions.

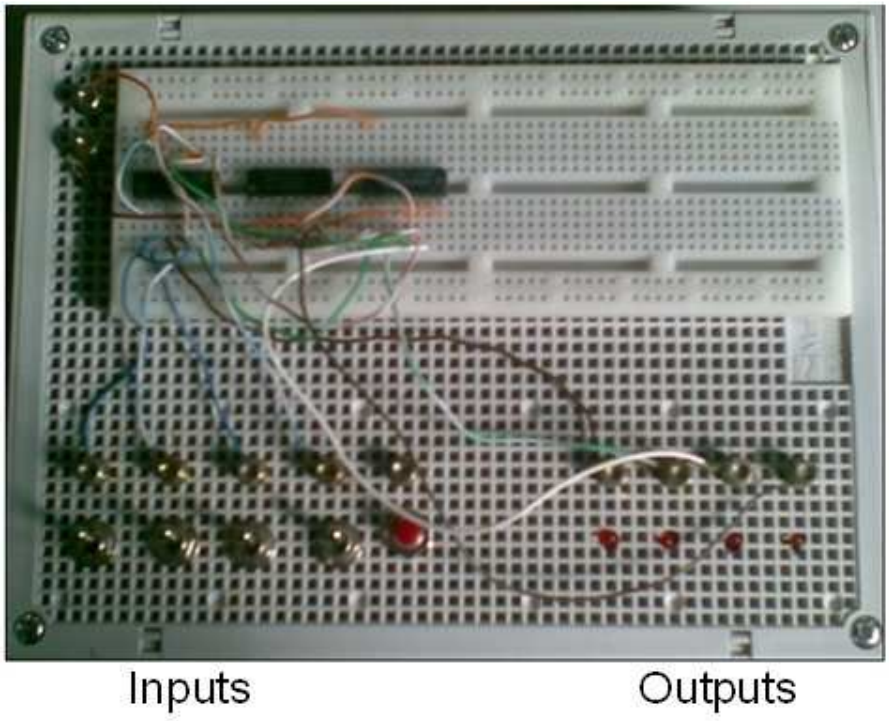


Figure 6.1 Binary Multiplier Circuit by Conventional Logic ICs

```
266         when 3 => DIN(15 downto 8) <= SW(7 downto 4) * SW(3 downto 0);
```

Figure 6.2 Execute the Multiplier Circuit by VHDL code

6.2 Conclusion

This work explained and given glance, about Programmable Logic devices their important and applications, specially FPGAs technology that used now in several areas and how programmed it by HDL language. This work explained also how written and execute several logic circuits by VHDL language that used in micro logic control applications, then implementation and verification of them on FPGA chip.

Hence, can write much logic functions by VHDL code and download these functions inside of FPGA chip rather than use several logic ICs, several external conductors, PCB size, the time, and the errors possibility, so the FPGA technology reduce the size of hardware and external conductors that used specially in big designs that required big size of PCB board.

From here, conclude the FPGA technology better alternative to other technologies such as ASIC and CPLD.

7. References

[1] Karen Parnell and Nick Mehta . *Introduction to Programmable Logic* . 2004.<http:// www.xilinx.com>

[2] *Complex Programmable Logic Devices* [0nline] 28/02/2010 < http:// www.wikipedia.org/wiki/complex_programmable_logic_device>.

[3] *Field Programmable gate array* [online]. last update Jun 2009. 04/07/2009 <http://www. wikipedia.org/wiki/fpga>.

[4] Andraka Consulting Group, inc. *FPGA Basics* [online]. Last update March 2007. 04/06/2009 <http://www.andraka.com/whatisan.htm>.

[5] Xilinx, Inc. *Spartan -3 FPGA Family Data sheet* [online]. Jun 2008. 01/12/2009 <http://www.xilinx.com>

[6] Jena P. Nicolle. *fpga4fun.com* [online].last update September 2009. 10/11/2009 <http://www.fpga4fun.com/FPGAinfo2.html>

[7] jameel Alkateeb. *Programmable logic devices* [online]. 2002-2005. 30/11/2009 <http:// www.handasarabia.org/phpBB2/viewtopic.php?t=3>

[8] *VHDL* [online]. Last update March 2010. 12/04/2010 <http://www.wikipedia.org/wiki/vhdl>.

[9] Jan Van der Spiegel. *VHDL Tutotial* [online]. Last update August 2006. 28/03/2010 <http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html#_Toc526061343>.

[10] Volnei A.pedroni. *Circuit design with VHDL*. 2004. ISBN 0-262-1622-5

[11] Rogr Tokahaem. *Digital Electronics*. 2001. ISBN 9953-3-0021-6

[12] Hoseen Almgbob. *Digital Technologies*. 2002-2003.

[13] Deab Mohamed Goneem. *Digital and Electronic Logic Circuit*. second edition 1999, ELGA company.

[14] Wayne Storr. *Electronics-Tutorials* [online]. Last update April 2010. 12/04/2010 <http://www.electronics-tutorials.ws/combination/comb_8.html>.

[15] *VHDL* [online]. Last update September 2009. 06/10/2009 <http://www.ar.wikipedia.org/wiki/%D9%81%D9%8A_%D8%A5%D8%AA%D8%B4_%D8%AF%D9%8A_%D8%A5%D9%84#.D8.A7.D9.84.D9.86.D8.A7.D8.AE.D8.A8>.

[16] Darren Wiessner. *Designing Synchronous Counter* [online]. 01/12/2009 <http:// http://ftp.csci.csusb.edu/schubert/tutorials/csci310/f03/dw4bit.pdf>.

[17] Xilinx, Inc. *Using Look -Up table as distributed RAM* [online]. March 2005. 15/03/2010 <http://www.xilinx.com>

[18] Ibrahim Kamal. *Tutorial De-bouncing Circuits* [online]. Last update May 2008. 21/11/2009. <http://www.ikalogic.com/debouncing.php>

[19] Wayne Storr. *Electronic-Tutorials* [online]. Last update November 2009. 27/11/2009.<http:// www.electronics-tutorials.ws/combination/comb_4.html>

[20] *Electronics Lab* [online]. 2002/2009. 03/01/2010. <http://www.electronics-lab.com/projects/oscillator_timers/014/index.html>

[21] Xilinx, Inc. *Transparent Data Latch* [online]. 01/12/ 2009 <http://www.xilinx.com>

[22] Xilinx, Inc. *OFDDRCPE* [online] 28/02/2010. <http://www.toolbox.xilinx.com/docsan/xilinx7/books/data/docs/lib/lib0229_285.html>.

[23] *Introduction to Accumulators and FPGAs* [online]. 15/03/2010. <http:// www.ece.unm.edu/vhdl/Labs2007/fall07/lab06/lab07_lecture.pdf>

[24] David Fritz. *Spartan 3 FPGA tutorial* [online]. Jun 2005 Oklahoma State University. 01/11/2009.
< http://www.repository.gunadarma.ac.id:8000/FPGA_Tutorial_129.pdf>

[25] Xilinx, Inc. *Spartan-3 Starter Kit Board User Guide* [online]. April 2004. 01/04/2010. <http://www.xilinx.com>

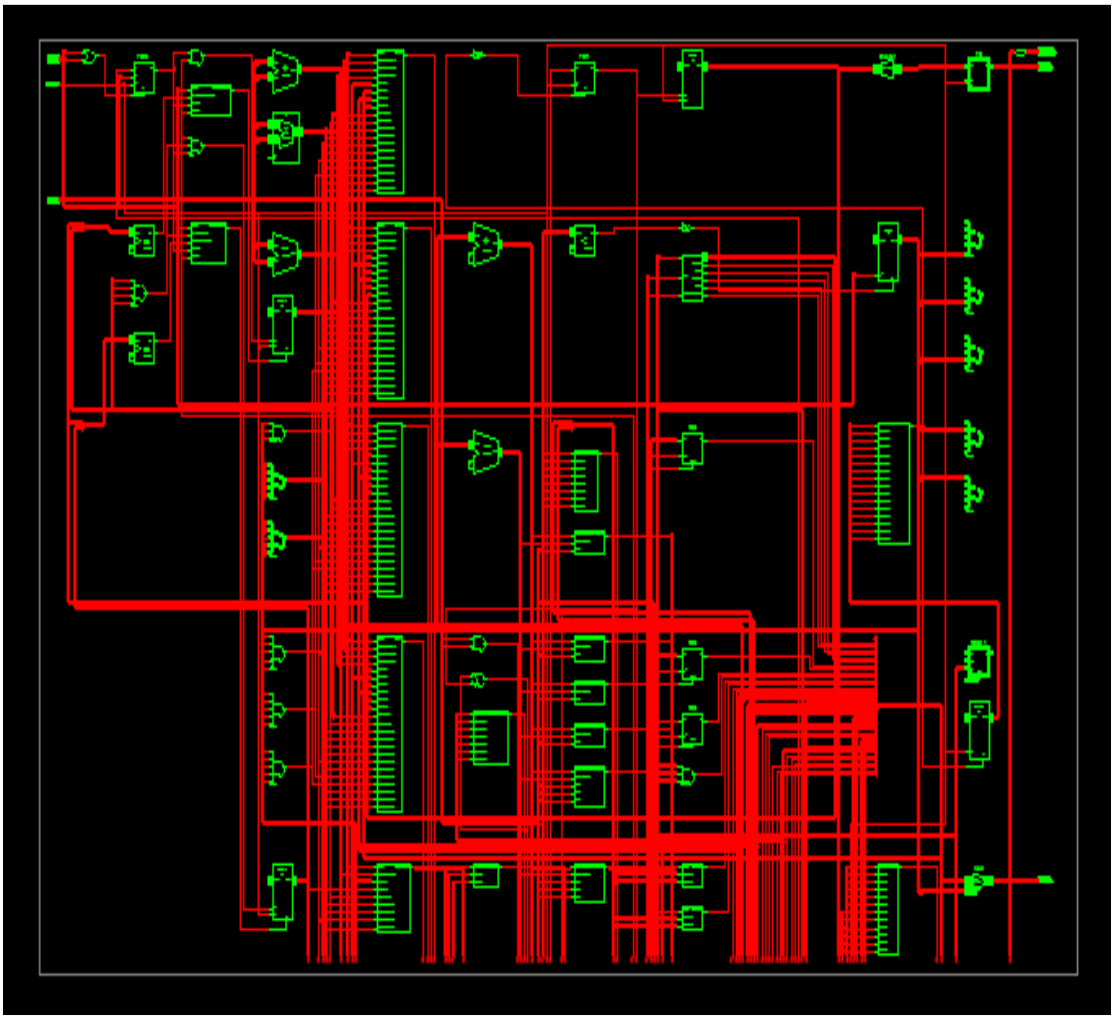
[26] Kasik, V.; , *"FPGA-Powered Embedded Vector Graphics,"* Mobile Ubiquitous Computing, Systems, Services and Technologies, 2008. UBICOMM '08. The Second International Conference on , Sept. 29 2008-Oct. 4 2008 doi: 10.1109/UBICOMM.2008.46.
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4641372&isnumber=4641296>

[27] Xilinx, Inc. *Xilinx Constraints Editor* [online]. 19/03/2010. <http://www.xilinx.com/itp/xilinx4/data/docs/cgd/entry8.html>

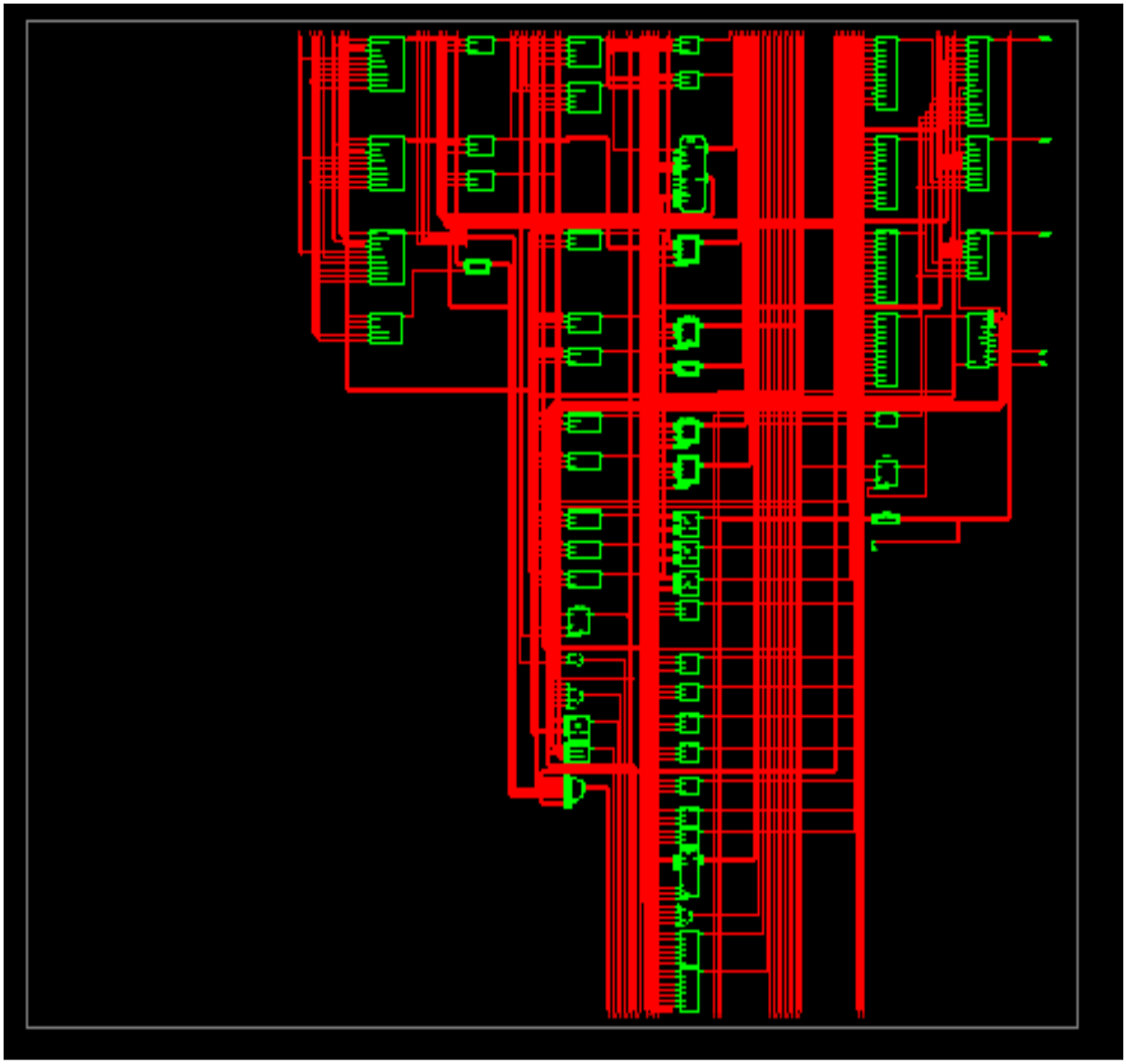
[28] Douglas L. Perry. *VHDL Programmable by Example*. fourth edition, 2004.

Appendix 1

RTL Schematic of the Logic Emulator Design



RTL Schematic of the Logic Emulator Design



RTL Schematic of the Logic Emulator Design